OptiML Documentation

Release 0.1

Stanford PPL

November 08, 2016

Contents

Contents:

CHAPTER 1

Introduction

<stub>

This document was auto-generated using Sphinx. For corrections, post an issue on GitHub Issues .

CHAPTER 2

Type Classes

2.1 Arith

<auto-generated stub>

2.1.1 Infix methods

```
def *(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def +(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def -(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def /(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def abs(x: Rep[T]): Rep[T]
```

```
def empty(): Rep[T]
```

```
def exp(x: Rep[T]): Rep[T]
```

```
def log(x: Rep[T]): Rep[T]
```

```
def zero(x: Rep[T]): Rep[T]
```

2.2 Bufferable

<auto-generated stub>

2.2.1 Infix methods

```
def mutable(x: Rep[T]): Rep[T]
```

```
def size(x: Rep[T]): Rep[Int]
```

```
def write(x: Rep[T], y: Rep[T]): Rep[Unit]
```

2.3 HasMinMax

<auto-generated stub>

2.3.1 Infix methods

```
def max(): Rep[T]
```

```
def min(): Rep[T]
```

2.4 Ordering

<auto-generated stub>

2.4.1 Infix methods

```
def !=(x: Rep[A], y: Rep[B]): Rep[Boolean]
```

```
def <(x: Rep[A], y: Rep[A]): Rep[Boolean]
```

```
def <=(x: Rep[A], y: Rep[A]): Rep[Boolean]
```

```
def >(x: Rep[A], y: Rep[A]): Rep[Boolean]
```

```
def >=(x: Rep[A], y: Rep[A]): Rep[Boolean]
```

```
def max(x: Rep[A], y: Rep[A]): Rep[A]
```

```
def min(x: Rep[A], y: Rep[A]): Rep[A]
```

2.4.2 Related methods

```
def __equal(x: Rep[A], y: Rep[B]): Rep[Boolean]
```

2.5 Shape

<auto-generated stub>

2.5.1 Infix methods

```
def apply(x: Rep[S], y: Rep[Int]): Rep[Tup2[Int,:doc:int]]
```

```
def contains(x: Rep[S], y: Rep[Int], z: Rep[Int]): Rep[Boolean]
```

```
def size(x: Rep[S]): Rep[Int]
```

2.6 Stringable

<auto-generated stub>

2.6.1 Infix methods

```
def makeStr(x: Rep[T]): Rep[String]
```

2.7 TrainingSetLike

<auto-generated stub>

2.7.1 Infix methods

```
def dot(x: Rep[TS[D,L]], y: Rep[Int], z: Rep[DenseVector[D]])(implicit ev0: Arith[D],ev1: N
```

```
def getCols(x: Rep[TS[D,L]], y: Rep[IndexVector])(implicit ev0: Manifest[TS[D,L]]): Rep[TS
```

```
def getRows(x: Rep[TS[D,L]], y: Rep[IndexVector])(implicit ev0: Manifest[TS[D,L]]): Rep[TS
```

```
def labels(x: Rep[TS[D,L]])(implicit ev0: Manifest[TS[D,L]]): Rep[DenseVector[L]]
```

```
def numFeatures(x: Rep[TS[D,L]])(implicit ev0: Manifest[TS[D,L]]): Rep[Int]
```

```
def numSamples(x: Rep[TS[D,L]])(implicit ev0: Manifest[TS[D,L]]): Rep[Int]
```

```
def times(x: Rep[TS[D,L]], y: Rep[Int], z: Rep[DenseVector[D]])(implicit ev0: Arith[D],ev1
```

```
def timesScalar(x: Rep[TS[D,L]], y: Rep[Int], z: Rep[D])(implicit ev0: Arith[D],ev1: Manife
```

CHAPTER 3

Data Structures

3.1 BinaryFeature

<auto-generated stub>

3.1.1 Static methods

```
def apply(default: Rep[Boolean] = false): Rep[BinaryFeature]
```

3.1.2 Infix methods

```
def apply(y: Rep[String]): Rep[Double]
```

```
def default(): Rep[Boolean]
```

3.2 CSRGraph

<auto-generated stub>

3.2.1 Static methods

```
def apply(numNodes: Rep[Int], numEdges: Rep[Int], nodes: Rep[ForgeArray[Int]], edges: Rep[I
```

```
def apply(nodes: Rep[DenseVector[Int]], edges: Rep[DenseVector[Int]]): Rep[CSRGraph]
```

3.2.2 Infix methods

```
def deepcopy(): Rep[CSRGraph]
```

```
def edges(): Rep[DenseVectorView[Int]]
```

```
def ngbrEdges(y: Rep[Int]): Rep[IndexVector]
```

```
def ngbrNodes(y: Rep[Int]): Rep[DenseVectorView[Int]]
```

```
def ngbrs(y: Rep[Int]): Rep[DenseVector[CSRNgbr]]
```

```
def nodes(): Rep[DenseVectorView[Int]]
```

```
def numEdges(): Rep[Int]
```

```
def numNodes(): Rep[Int]
```

3.3 CSRNgbr

<auto-generated stub>

3.3.1 Static methods

```
def apply(edgeId: Rep[Int], nodeId: Rep[Int]): Rep[CSRNgbr]
```

3.3.2 Infix methods

```
def edge(): Rep[Int]
```

```
def node(): Rep[Int]
```

3.4 Complex

<auto-generated stub>

3.4.1 Static methods

```
def apply(x: Rep[Double], y: Rep[Double]): Rep[Complex]
```

3.4.2 Infix methods

```
def *(y: Rep[Complex]): Rep[Complex]
```

```
def +(y: Rep[Complex]): Rep[Complex]
```

```
def -(y: Rep[Complex]): Rep[Complex]
```

```
def /(y: Rep[Complex]): Rep[Complex]
```

```
def abs(): Rep[Complex]
```

```
def conj(): Rep[Complex]
```

```
def exp(): Rep[Complex]
```

```
def imag(): Rep[Double]
```

```
def log(): Rep[Complex]
```

```
def real(): Rep[Double]
```

3.4.3 Related methods

```
def __equal(self: Rep[Complex], y: Rep[Complex]): Rep[Boolean]
```

3.5 ComputeStream

<auto-generated stub>

3.5.1 Static methods

```
def apply(numRows: Rep[Int], numCols: Rep[Int])(func: (Rep[Int],Rep[Int]) => Rep[T]): Rep[
```

3.5.2 Infix methods

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[T]
```

```
def foreach(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def foreachRow(y: (Rep[DenseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def numCols(): Rep[Int]
```

```
def numRows(): Rep[Int]
```

3.6 ContinuousFeature

<auto-generated stub>

3.6.1 Static methods

```
def apply(default: Rep[Double] = unit(0.0209), min: Rep[Double] = math_ninf(), max: Rep[Dou
```

3.6.2 Infix methods

```
def apply(y: Rep[String]): Rep[Double]
```

```
def default(): Rep[Double]
```

```
def max(): Rep[Double]
```

```
def min(): Rep[Double]
```

3.7 DDFGFEdge

<auto-generated stub>

3.7.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Int], z: Rep[Int], v: Rep[Boolean], w: Rep[Int]): Rep[DDFGFE
```

3.7.2 Infix methods

```
def equalPredicate(): Rep[Int]
```

```
def factorId(): Rep[Int]
```

```
def isPositive(): Rep[Boolean]
```

```
def position(): Rep[Int]
```

```
def variableId(): Rep[Int]
```

3.8 DDFGFFactor

<auto-generated stub>

3.8.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Int], z: Rep[Int], v: Rep[Int]): Rep[DDFGFFactor]
```

3.8.2 Infix methods

```
def edgeCount(): Rep[Int]
```

```
def factorFunction(): Rep[Int]
```

```
def factorId(): Rep[Int]
```

```
def weightId(): Rep[Int]
```

3.9 DDFGFVariable

<auto-generated stub>

3.9.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Boolean], z: Rep[Double], v: Rep[Int], w: Rep[Int], a: Rep[I
```

3.9.2 Infix methods

```
def cardinality(): Rep[Int]
```

```
def dataType(): Rep[Int]
```

```
def edgeCount(): Rep[Int]
```

```
def initialValue(): Rep[Double]
```

```
def isEvidence(): Rep[Boolean]
```

```
def variableId(): Rep[Int]
```

3.10 DDFGFWeight

<auto-generated stub>

3.10.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Boolean], z: Rep[Double]): Rep[DDFGFWeight]
```

3.10.2 Infix methods

```
def initialValue(): Rep[Double]
```

```
def isFixed(): Rep[Boolean]
```

```
def weightId(): Rep[Int]
```

3.11 DecisionTree

<auto-generated stub>

3.11.1 Static methods

```
def deserialize(encodedTree: Rep[String]): Rep[DecisionTree]
```

3.11.2 Infix methods

```
def addNode(parent: Rep[Int], isLeft: Rep[Boolean], isLeaf: Rep[Boolean], feature: Rep[Int]
```

```
def capacity(): Rep[Int]
```

```
def feature(): Rep[ForgeArray[Int]]
```

```
def impurity(): Rep[ForgeArray[Double]]
```

```
def isLeaf(): Rep[ForgeArray[Boolean]]
```

```
def leftChildren(): Rep[ForgeArray[Int]]
```

```
def numNodeSamples(): Rep[ForgeArray[Int]]
```

```
def numNodes(): Rep[Int]
```

```
def pprint(): Rep[Unit]
```

```
def predict(testPt: Rep[DenseVector[Double]]): Rep[Tup2[Double,:doc:double]]
```

```
def prob(): Rep[ForgeArray[Double]]
```

```
def rightChildren(): Rep[ForgeArray[Int]]
```

```
 def serialize(): Rep[String]
```

```
For now we use a simple text-format to persist the tree. Eventually, we probably will want
```

```
def threshold(): Rep[ForgeArray[Double]]
```

```
def value(): Rep[ForgeArray[Double]]
```

3.11.3 Related methods

```
def dtree(trainingSet: Rep[DenseTrainingSet[Double,:doc:double]], maxDepth: Rep[Int] = unit
```

3.12 DenseMatrix

<auto-generated stub>

3.12.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def apply(x: Rep[ForgeArray[T]], y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def apply(x: Rep[DenseVector[DenseVector[T]]]): Rep[DenseMatrix[T]]
```

```
def apply(x: Rep[DenseVector[DenseVectorView[T]]]): Rep[DenseMatrix[T]]
```

```
def apply(x: Rep[DenseVector[T]]*): Rep[DenseMatrix[T]]
```

```
def block(x: Rep[DenseVector[DenseMatrix[T]]]*): Rep[DenseMatrix[T]]
```

```
def diag(x: Rep[Int], y: Rep[DenseVector[T]]): Rep[DenseMatrix[T]]
```

```
def identity(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def identity(x: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def ones(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def onesf(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def rand(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def randf(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def randn(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def randnf(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def zeros(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def zerosf(x: Rep[Int], y: Rep[Int]): Rep[DenseMatrix[Float]]
```

3.12.2 Infix methods

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def *(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[Float]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *:*(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def *:*(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *:*(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *:*(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def *:*(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *:*(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *:*(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def *:*(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def *:*(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def *=(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def *=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
 def +(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

Math

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def +(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def +(y: Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def +(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[Float]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
 def +=(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

Math

```
def +=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def -(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def -(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[Float]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -=(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def -=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def /(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def /(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[Int]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[Int]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[Float]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def /(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def /(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def /(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def /=(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def /=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def :<(y: Rep[DenseMatrix[T]])(implicit ev0: Ordering[T]): Rep[DenseMatrix[Boolean]]
```

```
def :>(y: Rep[DenseMatrix[T]])(implicit ev0: Ordering[T]): Rep[DenseMatrix[Boolean]]
```

```
def <<(y: Rep[DenseVector[T]]): Rep[DenseMatrix[T]]
```

```
def <<(y: Rep[DenseMatrix[T]]): Rep[DenseMatrix[T]]
```

```
def <<=(y: Rep[DenseVector[T]]): Rep[Unit]
```

```
def <<=(y: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def <<|(y: Rep[DenseVector[T]]): Rep[DenseMatrix[T]]
```

```
def <<|(y: Rep[DenseMatrix[T]]): Rep[DenseMatrix[T]]
```

```
def <<|=(y: Rep[DenseVector[T]]): Rep[Unit]
```

```
def <<|=(y: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
 def Clone(): Rep[DenseMatrix[T]]
```

Miscellaneous

```
def (y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[T]
```

```
 def apply(y: Rep[Int]): Rep[DenseVectorView[T]]
```

Accessors

```
def apply(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def apply(y: Rep[IndexVector], z: IndexWildcard): Rep[DenseMatrix[T]]
```

```
def apply(rows: Rep[IndexVector], cols: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def apply(y: IndexWildcard, z: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def colIndices(): Rep[IndexVector]
```

```
def count(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def diag(): Rep[DenseVector[T]]
```

```
def exp()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def filterCols(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[DenseMatrix[T]]
```

```
def filterRows(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[DenseMatrix[T]]
```

```
def findCols(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
def findRows(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
 def flattenToVector(): Rep[DenseVector[T]]
```

Conversions

```
def foreach(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def foreachCol(y: (Rep[DenseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def foreachRow(y: (Rep[DenseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def getCol(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def getCols(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def getRow(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def getRows(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def groupColsBy(y: (Rep[DenseVectorView[T]]) => Rep[K]): Rep[ForgeHashMap[K,:doc:densematr
```

```
 def groupRowsBy(y: (Rep[DenseVectorView[T]]) => Rep[K]): Rep[ForgeHashMap[K,:doc:densemat
```

Bulk

```
def indices(): Rep[IndexVector]
```

```
def insertAllCols(pos: Rep[Int], xs: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def insertAllRows(pos: Rep[Int], xs: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def insertCol(pos: Rep[Int], y: Rep[DenseVector[T]]): Rep[Unit]
```

```
def insertRow(pos: Rep[Int], y: Rep[DenseVector[T]]): Rep[Unit]
```

```
def log()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def makeDimsStr(): Rep[String]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
 def map(y: (Rep[T]) => Rep[R]): Rep[DenseMatrix[R]]
```

Bulk

```
def mapCols(y: (Rep[DenseVectorView[T]]) => Rep[DenseVector[R]]): Rep[DenseMatrix[R]]
```

```
def mapColsToVector(y: (Rep[DenseVectorView[T]]) => Rep[R]): Rep[DenseVector[R]]
```

```
def mapRows(y: (Rep[DenseVectorView[T]]) => Rep[DenseVector[R]]): Rep[DenseMatrix[R]]
```

```
def mapRowsToVector(y: (Rep[DenseVectorView[T]]) => Rep[R]): Rep[DenseVector[R]]
```

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def maxCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def maxIndex()(implicit ev0: Ordering[T]): Rep[Tup2[Int,:doc:int]]
```

```
def maxRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def minCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def minIndex()(implicit ev0: Ordering[T]): Rep[Tup2[Int,:doc:int]]
```

```
 def minRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
Ordering
```

```
def mutable(): Rep[DenseMatrix[T]]
```

```
def mview(startRow: Rep[Int], endRow: Rep[Int], startCol: Rep[Int], endCol: Rep[Int]): Rep
```

```
def numCols(): Rep[Int]
```

```
 def numRows(): Rep[Int]
```

```
Accessors
```

```
 def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

```
Miscellaneous
```

```
def prod()(implicit ev0: Arith[T]): Rep[T]
```

```
def reduce(y: (Rep[T],Rep[T]) => Rep[T])(implicit ev0: Arith[T]): Rep[T]
```

```
def reduceCols(y: (Rep[DenseVector[T]],Rep[DenseVector[T]]) => Rep[DenseVector[T]])(implici
```

```
def reduceRows(y: (Rep[DenseVector[T]],Rep[DenseVector[T]]) => Rep[DenseVector[T]])(implici
```

```
def removeCol(pos: Rep[Int]): Rep[Unit]
```

```
def removeCols(pos: Rep[Int], num: Rep[Int]): Rep[Unit]
```

```
def removeRow(pos: Rep[Int]): Rep[Unit]
```

```
def removeRows(pos: Rep[Int], num: Rep[Int]): Rep[Unit]
```

```
def replicate(y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def rowIndices(): Rep[IndexVector]
```

```
def size(): Rep[Int]
```

```
def slice(startRow: Rep[Int], endRow: Rep[Int], startCol: Rep[Int], endCol: Rep[Int]): Rep
```

```
def sliceCols(start: Rep[Int], end: Rep[Int]): Rep[DenseMatrixView[T]]
```

```
def sliceRows(start: Rep[Int], end: Rep[Int]): Rep[DenseMatrixView[T]]
```

```
def sortColsBy(y: (Rep[DenseVectorView[T]]) => Rep[B])(implicit ev0: Ordering[B]): Rep[Dens
```

```
 def sortRowsBy(y: (Rep[DenseVectorView[T]]) => Rep[B])(implicit ev0: Ordering[B]): Rep[Den
Ordering
```

```
def stddev()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
def sumCols()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def sumRows()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def t(): Rep[DenseMatrix[T]]
```

```
def toArray(): Rep[ForgeArray[T]]
```

```
 def toBoolean()(implicit ev0: (Rep[T]) => Rep[Boolean]): Rep[DenseMatrix[Boolean]]
Conversions
```

```
def toDouble()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def toFloat()(implicit ev0: (Rep[T]) => Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def toInt()(implicit ev0: (Rep[T]) => Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def toSparse(): Rep[SparseMatrix[T]]
```

```
def toString(): Rep[String]
```

```
def tril()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def trim(): Rep[Unit]
```

```
def triu()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def unary_-(): Rep[DenseMatrix[Int]]
```

```
def unary_-(): Rep[DenseMatrix[Float]]
```

```
def unary_-(): Rep[DenseMatrix[Double]]
```

```
def update(y: Rep[Int], z: Rep[Int], v: Rep[T]): Rep[Unit]
```

```
def update(y: Rep[Int], z: Rep[DenseVector[T]]): Rep[Unit]
```

```
def update(y: Rep[Int], z: Rep[DenseVectorView[T]]): Rep[Unit]
```

```
def update(y: Rep[IndexVector], z: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def updateCol(y: Rep[Int], z: Rep[DenseVector[T]]): Rep[Unit]
```

```
def updateCol(y: Rep[Int], z: Rep[DenseVectorView[T]]): Rep[Unit]
```

```
def updateCols(y: Rep[IndexVector], z: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def updateRow(y: Rep[Int], z: Rep[DenseVector[T]]): Rep[Unit]
```

```
def updateRow(y: Rep[Int], z: Rep[DenseVectorView[T]]): Rep[Unit]
```

```
def updateRows(y: Rep[IndexVector], z: Rep[DenseMatrix[T]]): Rep[Unit]
```

```
def variance()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def vview(start: Rep[Int], stride: Rep[Int], length: Rep[Int], isRow: Rep[Boolean]): Rep[De
```

```
def zip(y: Rep[DenseMatrix[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseMatrix[R]]
```

3.12.3 Implicit methods

```
def dist(x: Rep[DenseMatrix[Double]], y: Rep[DenseMatrix[Double]]): Rep[Double]
```

3.12.4 Related methods

```
def __equal(self: Rep[DenseMatrix\[T\]], y: Rep[DenseMatrix[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[DenseMatrix\[T\]], y: Rep[DenseMatrixView[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[DenseMatrix\[T\]], y: Rep[SparseMatrix[T]]): Rep[Boolean]
```

```
def densematrix_fromarray(x: Rep[ForgeArray[T]], y: Rep[Int], z: Rep[Int]): Rep[DenseMatri
```

```
def densematrix_fromfunc(x: Rep[Int], y: Rep[Int], z: (Rep[Int],Rep[Int]) => Rep[T]): Rep[I
```

```
 def densematrix_raw_apply(self: Rep[DenseMatrix\[T\]], y: Rep[Int]): Rep[T]
```

Required for parallel collection

```
def densematrix_raw_update(self: Rep[DenseMatrix\[T\]], y: Rep[Int], z: Rep[T]): Rep[Unit]
```

```
def diag(x: Rep[DenseMatrix[T]]): Rep[DenseVector[T]]
```

```
def dist(x: Rep[DenseMatrix[Double]], y: Rep[DenseMatrix[Double]], z: DistanceMetric): Rep
```

```
def norm(x: Rep[DenseMatrix[Double]]): Rep[Double]
```

```
def norm(x: Rep[DenseMatrix[Double]], y: NormId): Rep[Double]
```

```
def tril(x: Rep[DenseMatrix[T]]): Rep[DenseMatrix[T]]
```

```
def triu(x: Rep[DenseMatrix[T]]): Rep[DenseMatrix[T]]
```

3.13 DenseMatrixView

<auto-generated stub>

3.13.1 Static methods

```
def apply(x: Rep[ForgeArray[T]], y: Rep[Int], z: Rep[Int], v: Rep[Int], w: Rep[Int], a: Rep
```

3.13.2 Infix methods

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *:*(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
 def +(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

Math

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def +(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def Clone(): Rep[DenseMatrix[T]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[T]
```

```
 def apply(y: Rep[Int]): Rep[DenseVectorView[T]]
```

Accessors

```
def apply(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def apply(y: Rep[IndexVector], z: IndexWildcard): Rep[DenseMatrix[T]]
```

```
def apply(rows: Rep[IndexVector], cols: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def apply(y: IndexWildcard, z: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def colIndices(): Rep[IndexVector]
```

```
def count(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def exp()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def filterCols(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[DenseMatrix[T]]
```

```
def filterRows(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[DenseMatrix[T]]
```

```
def findCols(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
def findRows(y: (Rep[DenseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
def foreach(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def foreachCol(y: (Rep[DenseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def foreachRow(y: (Rep[DenseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def getCol(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def getCols(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def getRow(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def getRows(y: Rep[IndexVector]): Rep[DenseMatrix[T]]
```

```
def indices(): Rep[IndexVector]
```

```
def log()(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def makeDimsStr(): Rep[String]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
 def map(y: (Rep[T]) => Rep[R]): Rep[DenseMatrix[R]]
Bulk
```

```
def mapCols(y: (Rep[DenseVectorView[T]]) => Rep[DenseVector[R]]): Rep[DenseMatrix[R]]
```

```
def mapColsToVector(y: (Rep[DenseVectorView[T]]) => Rep[R]): Rep[DenseVector[R]]
```

```
def mapRows(y: (Rep[DenseVectorView[T]]) => Rep[DenseVector[R]]): Rep[DenseMatrix[R]]
```

```
def mapRowsToVector(y: (Rep[DenseVectorView[T]]) => Rep[R]): Rep[DenseVector[R]]
```

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def maxCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def maxIndex()(implicit ev0: Ordering[T]): Rep[Tup2[Int,:doc:int]]
```

```
def maxRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def minCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
```

```
def minIndex()(implicit ev0: Ordering[T]): Rep[Tup2[Int,:doc:int]]
```

```
 def minRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[DenseVector[T]]
Ordering
```

```
def mutable(): Rep[DenseMatrix[T]]
```

```
def mview(startRow: Rep[Int], endRow: Rep[Int], startCol: Rep[Int], endCol: Rep[Int]): Rep
```

```
def numCols(): Rep[Int]
```

```
def numRows(): Rep[Int]
```

```
 def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

Miscellaneous

```
def prod()(implicit ev0: Arith[T]): Rep[T]
```

```
def reduce(y: (Rep[T],Rep[T]) => Rep[T])(implicit ev0: Arith[T]): Rep[T]
```

```
def reduceCols(y: (Rep[DenseVector[T]],Rep[DenseVector[T]]) => Rep[DenseVector[T]])(implici
```

```
def reduceRows(y: (Rep[DenseVector[T]],Rep[DenseVector[T]]) => Rep[DenseVector[T]])(implici
```

```
def replicate(y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def rowIndices(): Rep[IndexVector]
```

```
def size(): Rep[Int]
```

```
def slice(startRow: Rep[Int], endRow: Rep[Int], startCol: Rep[Int], endCol: Rep[Int]): Rep
```

```
def sliceCols(start: Rep[Int], end: Rep[Int]): Rep[DenseMatrixView[T]]
```

```
def sliceRows(start: Rep[Int], end: Rep[Int]): Rep[DenseMatrixView[T]]
```

```
def stddev()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
def sumCols()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def sumRows()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def t(): Rep[DenseMatrix[T]]
```

```
 def toBoolean()(implicit ev0: (Rep[T]) => Rep[Boolean]): Rep[DenseMatrix[Boolean]]
```

Conversions

```
def toDense(): Rep[DenseMatrix[T]]
```

```
def toDouble()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[DenseMatrix[Double]]
```

```
def toFloat()(implicit ev0: (Rep[T]) => Rep[Float]): Rep[DenseMatrix[Float]]
```

```
def toInt()(implicit ev0: (Rep[T]) => Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def toString(): Rep[String]
```

```
def variance()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def vview(start: Rep[Int], stride: Rep[Int], length: Rep[Int], isRow: Rep[Boolean]): Rep[De
```

```
def zip(y: Rep[DenseMatrix[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseMatrix[R]]
```

3.13.3 Implicit methods

```
def chainViewToDenseOps(self: Rep[DenseMatrixView\[T\]]): DenseMatrixDenseMatrixOpsCls[T]
```

```
def viewToDense(self: Rep[DenseMatrixView\[T\]]): Rep[DenseMatrix[T]]
```

3.13.4 Related methods

```
def __equal(self: Rep[DenseMatrixView\[T\]], y: Rep[DenseMatrix[T]]): Rep[Boolean]
```

```
def densematrixview_raw_apply(self: Rep[DenseMatrixView\[T\]], y: Rep[Int]): Rep[T]
```

```
def norm(x: Rep[DenseMatrixView[Double]]): Rep[Double]
```

```
def norm(x: Rep[DenseMatrixView[Double]], y: NormId): Rep[Double]
```

3.14 DenseTrainingSet

<auto-generated stub>

3.14.1 Static methods

```
def apply(x: Rep[DenseMatrix[D]], y: Rep[DenseVector[L]]): Rep[DenseTrainingSet[D,L]]
```

3.14.2 Infix methods

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[D]
```

```
def apply(y: Rep[Int]): Rep[DenseVectorView[D]]
```

```
def data(): Rep[DenseMatrix[D]]
```

```
def labels(): Rep[DenseVector[L]]
```

```
def numFeatures(): Rep[Int]
```

```
def numSamples(): Rep[Int]
```

3.15 DenseVector

<auto-generated stub>

3.15.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Boolean]): Rep[DenseVector[T]]
```

```
def apply(x: Rep[T]*): Rep[DenseVector[T]]
```

```
def apply(x: Rep[ForgeArray[T]], isRow: Rep[Boolean] = true): Rep[DenseVector[T]]
```

```
def flatten(pieces: Rep[DenseVector[DenseVector[T]]]): Rep[DenseVector[T]]
```

```
def ones(x: Rep[Int]): Rep[DenseVector[Double]]
```

```
def onesf(x: Rep[Int]): Rep[DenseVector[Float]]
```

```
def rand(x: Rep[Int]): Rep[DenseVector[Double]]
```

```
def randf(x: Rep[Int]): Rep[DenseVector[Float]]
```

```
def uniform(start: Rep[Double], step_size: Rep[Double], end: Rep[Double], isRow: Rep[Boolea
```

```
def zeros(x: Rep[Int]): Rep[DenseVector[Double]]
```

```
def zerosf(x: Rep[Int]): Rep[DenseVector[Float]]
```

3.15.2 Infix methods

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[Int]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[Int]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[Float]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseVector[Double]]
```

```
def **(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *=(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def *=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def *=(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
 def +(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

Math

```
def +(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[Int]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[Int]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[Float]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
 def +=(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

Math

```
def +=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def +=(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def -(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[Int]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[Int]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[Float]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -=(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def -=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def -=(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def /(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[Int]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[Float]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[Int]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[Float]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[Double]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def /(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def /(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def /=(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def /=(y: Rep[T])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def /=(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[Unit]
```

```
def :<(y: Rep[DenseVector[T]])(implicit ev0: Ordering[T]): Rep[DenseVector[Boolean]]
```

```
def :>(y: Rep[DenseVector[T]])(implicit ev0: Ordering[T]): Rep[DenseVector[Boolean]]
```

```
def <<(y: Rep[T]): Rep[DenseVector[T]]
```

```
def <<(y: Rep[DenseVector[T]]): Rep[DenseVector[T]]
```

```
def <<=(y: Rep[T]): Rep[Unit]
```

```
def <<=(y: Rep[DenseVector[T]]): Rep[Unit]
```

```
def Clone(): Rep[DenseVector[T]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def apply(y: Rep[Int]): Rep[T]
```

```
def apply(y: Rep[IndexVector]): Rep[DenseVector[T]]
```

```
def clear(): Rep[Unit]
```

```
def contains(y: Rep[T]): Rep[Boolean]
```

```
def copyFrom(y: Rep[Int], z: Rep[DenseVector[T]]): Rep[Unit]
```

```
def count(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def distinct(): Rep[DenseVector[T]]
```

```
def drop(y: Rep[Int]): Rep[DenseVector[T]]
```

```
def exists(y: (Rep[T]) => Rep[Boolean]): Rep[Boolean]
```

```
def exp()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def filter(y: (Rep[T]) => Rep[Boolean]): Rep[DenseVector[T]]
```

```
def find(y: (Rep[T]) => Rep[Boolean]): Rep[IndexVector]
```

```
def first(): Rep[T]
```

```
def flatMap(y: (Rep[T]) => Rep[DenseVector[R]]): Rep[DenseVector[R]]
```

```
def forall(y: (Rep[T]) => Rep[Boolean]): Rep[Boolean]
```

```
def foreach(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def groupBy(y: (Rep[T]) => Rep[K], z: (Rep[T]) => Rep[V]): Rep[ForgeHashMap[K,:doc:denseve
```

```
 def groupByReduce(y: (Rep[T]) => Rep[K], z: (Rep[T]) => Rep[V], v: (Rep[V],Rep[V]) => Rep
```

```
Bulk
```

```
def histogram(): Rep[ForgeHashMap[T,:doc:int]]
```

```
 def indices(): Rep[IndexVector]
```

```
Accessors
```

```
def insert(y: Rep[Int], z: Rep[T]): Rep[Unit]
```

```
def insertAll(y: Rep[Int], z: Rep[DenseVector[T]]): Rep[Unit]
```

```
def intersect(y: Rep[DenseVector[T]]): Rep[DenseVector[T]]
```

```
def isEmpty(): Rep[Boolean]
```

```
def isRow(): Rep[Boolean]
```

```
def last(): Rep[T]
```

```
 def length(): Rep[Int]
```

Accessors

```
def log()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def makeStrWithDelim(delim: Rep[String])(implicit ev0: Stringable[T]): Rep[String]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
 def map(y: (Rep[T]) => Rep[R]): Rep[DenseVector[R]]
```

Bulk

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def maxIndex()(implicit ev0: Ordering[T]): Rep[Int]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def median()(implicit ev0: Numeric[T],ev1: Ordering[T]): Rep[T]
```

```
 def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

Ordering

```
def minIndex()(implicit ev0: Ordering[T]): Rep[Int]
```

```
def mt(): Rep[Unit]
```

```
def mutable(): Rep[DenseVector[T]]
```

```
def partition(pred: (Rep[T]) => Rep[Boolean]): Rep[Tup2[DenseVector[T],:doc:densevector[T]]
```

```
def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

```
def prefixSum()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def prod()(implicit ev0: Arith[T]): Rep[T]
```

```
def reduce(y: (Rep[T],Rep[T]) => Rep[T])(implicit ev0: Arith[T]): Rep[T]
```

```
def remove(y: Rep[Int]): Rep[Unit]
```

```
def removeAll(pos: Rep[Int], len: Rep[Int]): Rep[Unit]
```

```
def replicate(y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def scanLeft(zero: Rep[R])(z: (Rep[R],Rep[T]) => Rep[R]): Rep[DenseVector[R]]
```

```
def scanRight(zero: Rep[R])(z: (Rep[T],Rep[R]) => Rep[R]): Rep[DenseVector[R]]
```

```
def slice(start: Rep[Int], end: Rep[Int]): Rep[DenseVector[T]]
```

```
 def sort()(implicit ev0: Ordering[T]): Rep[DenseVector[T]]
```

Ordering

```
def sortBy(y: (Rep[T]) => Rep[B])(implicit ev0: Ordering[B]): Rep[DenseVector[T]]
```

```
def sortWithIndex()(implicit ev0: Ordering[T]): Tuple2[Rep[DenseVector[T]],Rep[IndexVector]
```

```
def stddev()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
 def t(): Rep[DenseVector[T]]
```

Miscellaneous

```
def take(y: Rep[Int]): Rep[DenseVector[T]]
```

```
 def toArray(): Rep[ForgeArray[T]]
```

Data exchange

```
 def toBoolean()(implicit ev0: (Rep[T]) => Rep[Boolean]): Rep[DenseVector[Boolean]]
```

Conversions

```
def toDense(): Rep[DenseVector[T]]
```

```
def toDouble()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[DenseVector[Double]]
```

```
def toFloat()(implicit ev0: (Rep[T]) => Rep[Float]): Rep[DenseVector[Float]]
```

```
def toInt()(implicit ev0: (Rep[T]) => Rep[Int]): Rep[DenseVector[Int]]
```

```
def toMat(): Rep[DenseMatrix[T]]
```

```
def toString(): Rep[String]
```

```
def trim(): Rep[Unit]
```

```
def unary_-(): Rep[DenseVector[Int]]
```

```
def unary_-(): Rep[DenseVector[Float]]
```

```
def unary_-(): Rep[DenseVector[Double]]
```

```
def update(i: Rep[Int], e: Rep[T]): Rep[Unit]
```

```
def update(indices: Rep[IndexVector], e: Rep[T]): Rep[Unit]
```

```
def update(indices: Rep[IndexVector], v: Rep[DenseVector[T]]): Rep[Unit]
```

```
def variance()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def zip(y: Rep[DenseVector[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

```
def zip(y: Rep[DenseVectorView[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

3.15.3 Implicit methods

```
def dist(x: Rep[DenseVector[Double]], y: Rep[DenseVector[Double]]): Rep[Double]
```

3.15.4 Related methods

```
def __equal(self: Rep[DenseVector\[T\]], y: Rep[DenseVector[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[DenseVector\[T\]], y: Rep[DenseVectorView[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[DenseVector\[T\]], y: Rep[IndexVector]): Rep[Boolean]
```

```
def __equal(self: Rep[DenseVector\[T\]], y: Rep[SparseVector[T]]): Rep[Boolean]
```

```
def densevector_fromarray(x: Rep[ForgeArray[T]], y: Rep[Boolean]): Rep[DenseVector[T]]
```

```
def densevector_fromfunc(x: Rep[Int], y: Rep[Boolean], z: (Rep[Int]) => Rep[T]): Rep[DenseV
```

```
def dist(x: Rep[DenseVector[Double]], y: Rep[DenseVector[Double]], z: DistanceMetric): Rep
```

```
def norm(x: Rep[DenseVector[Double]]): Rep[Double]
```

```
def norm(x: Rep[DenseVector[Double]], y: NormId): Rep[Double]
```

3.16 DenseVectorView

<auto-generated stub>

3.16.1 Static methods

```
def apply(x: Rep[ForgeArray[T]], y: Rep[Int], z: Rep[Int], v: Rep[Int], w: Rep[Boolean]): R
```

3.16.2 Infix methods

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def **(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def **(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
 def +(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

Math

```
def +(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[SparseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def Clone(): Rep[DenseVector[T]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def apply(y: Rep[Int]): Rep[T]
```

```
def apply(y: Rep[IndexVector]): Rep[DenseVector[T]]
```

```
def contains(y: Rep[T]): Rep[Boolean]
```

```
def count(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def distinct(): Rep[DenseVector[T]]
```

```
def drop(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def exists(y: (Rep[T]) => Rep[Boolean]): Rep[Boolean]
```

```
def exp()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def filter(y: (Rep[T]) => Rep[Boolean]): Rep[DenseVector[T]]
```

```
def find(y: (Rep[T]) => Rep[Boolean]): Rep[IndexVector]
```

```
def first(): Rep[T]
```

```
def flatMap(y: (Rep[T]) => Rep[DenseVector[R]]): Rep[DenseVector[R]]
```

```
def forall(y: (Rep[T]) => Rep[Boolean]): Rep[Boolean]
```

```
def foreach(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def histogram(): Rep[ForgeHashMap[T,:doc:int]]
```

```
 def indices(): Rep[IndexVector]
```

Accessors

```
def intersect(y: Rep[DenseVector[T]]): Rep[DenseVector[T]]
```

```
def isEmpty(): Rep[Boolean]
```

```
def isRow(): Rep[Boolean]
```

```
def last(): Rep[T]
```

```
def length(): Rep[Int]
```

```
def log()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def makeStrWithDelim(delim: Rep[String])(implicit ev0: Stringable[T]): Rep[String]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
 def map(y: (Rep[T]) => Rep[R]): Rep[DenseVector[R]]
```

Bulk

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def maxIndex()(implicit ev0: Ordering[T]): Rep[Int]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
 def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

Ordering

```
def minIndex()(implicit ev0: Ordering[T]): Rep[Int]
```

```
def mutable(): Rep[DenseVector[T]]
```

```
def partition(pred: (Rep[T]) => Rep[Boolean]): Rep[Tup2[DenseVector[T],:doc:densevector[T]]
```

```
def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

```
def prefixSum()(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def prod()(implicit ev0: Arith[T]): Rep[T]
```

```
def reduce(y: (Rep[T],Rep[T]) => Rep[T])(implicit ev0: Arith[T]): Rep[T]
```

```
def replicate(y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[T]]
```

```
def scanLeft(zero: Rep[R])(z: (Rep[R],Rep[T]) => Rep[R]): Rep[DenseVector[R]]
```

```
def scanRight(zero: Rep[R])(z: (Rep[T],Rep[R]) => Rep[R]): Rep[DenseVector[R]]
```

```
def slice(start: Rep[Int], end: Rep[Int]): Rep[DenseVectorView[T]]
```

```
def stddev()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
def take(y: Rep[Int]): Rep[DenseVectorView[T]]
```

```
 def toArray(): Rep[ForgeArray[T]]
```

Data exchange

```
 def toBoolean()(implicit ev0: (Rep[T]) => Rep[Boolean]): Rep[DenseVector[Boolean]]
```

Conversions

```
def toDense(): Rep[DenseVector[T]]
```

```
def toDouble()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[DenseVector[Double]]
```

```
def toFloat()(implicit ev0: (Rep[T]) => Rep[Float]): Rep[DenseVector[Float]]
```

```
def toInt()(implicit ev0: (Rep[T]) => Rep[Int]): Rep[DenseVector[Int]]
```

```
def toString(): Rep[String]
```

```
def variance()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def zip(y: Rep[DenseVector[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

```
def zip(y: Rep[DenseVectorView[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

3.16.3 Implicit methods

```
def chainViewToDenseOps(self: Rep[DenseVectorView\[T\]]): DenseVectorDenseVectorOpsCls[T]
```

```
def dist(x: Rep[DenseVectorView[Double]], y: Rep[DenseVectorView[Double]]): Rep[Double]
```

```
def viewToDense(self: Rep[DenseVectorView\[T\]]): Rep[DenseVector[T]]
```

3.16.4 Related methods

```
def __equal(self: Rep[DenseVectorView\[T\]], y: Rep[DenseVector[T]]): Rep[Boolean]
```

```
def dist(x: Rep[DenseVectorView[Double]], y: Rep[DenseVectorView[Double]], z: DistanceMetri
```

```
def norm(x: Rep[DenseVectorView[Double]]): Rep[Double]
```

```
def norm(x: Rep[DenseVectorView[Double]], y: NormId): Rep[Double]
```

3.17 DiscreteFeature

<auto-generated stub>

3.17.1 Static methods

```
def apply(x: Rep[String]*): Rep[DiscreteFeature]
```

3.17.2 Infix methods

```
def apply(y: Rep[String]): Rep[Double]
```

```
def indicator(y: Rep[String]): Rep[DenseVector[Double]]
```

```
def size(): Rep[Int]
```

3.18 FactorGraph

<auto-generated stub>

3.18.1 Static methods

```
def apply(v2f: Rep[CSRGraph], f2v: Rep[CSRGraph], weightValue: Rep[DenseVector[Double]], we
```

3.18.2 Infix methods

```
def deepcopy(): Rep[FactorGraph]
```

```
def edgeIsPositiveF2V(): Rep[DenseVector[Boolean]]
```

```
def f2v(): Rep[CSRGraph]
```

```
def factorFunction(): Rep[DenseVector[Int]]
```

```
def factorWeightIdx(): Rep[DenseVector[Int]]
```

```
def mutable(): Rep[FactorGraph]
```

```
def mutableVariables(): Rep[FactorGraph]
```

```
def mutableWeights(): Rep[FactorGraph]
```

```
def nonEvidenceVariables(): Rep[DenseVector[Int]]
```

```
def numEdges(): Rep[Int]
```

```
def numFactors(): Rep[Int]
```

```
def numVariables(): Rep[Int]
```

```
def numWeights(): Rep[Int]
```

```
def v2f(): Rep[CSRGraph]
```

```
def variableIsEvidence(): Rep[DenseVector[Boolean]]
```

```
def variableValue(): Rep[DenseVector[Boolean]]
```

```
def weightIsFixed(): Rep[DenseVector[Boolean]]
```

```
def weightValue(): Rep[DenseVector[Double]]
```

3.19 FileStream

<auto-generated stub>

3.19.1 Static methods

```
def apply(x: Rep[String]): Rep[FileStream]
```

3.19.2 Infix methods

```
def foreach(y: (Rep[String]) => Rep[Unit]): Rep[Unit]
```

```
def groupRowsBy(outTable: Rep[String], delim: Rep[String] = unit(``\s+''), trim: Rep[Boolea
```

```
def map(outFile: Rep[String], preserveOrder: Rep[Boolean] = false, chunkSize: Rep[Long] = 
```

```
def mapRows(outFile: Rep[String], inDelim: Rep[String] = unit(``\s+''), outDelim: Rep[Strin
```

```
def path(): Rep[String]
```

```
def processFileChunks(readFunc: (Rep[String],Rep[String]) => Rep[R], processFunc: (Rep[For
```

```
def reduce(zero: Rep[T])(func: (Rep[String]) => Rep[T])(rfunc: (Rep[T],Rep[T]) => Rep[T]):
```

3.19.3 Related methods

```
def getChunkByteSize(): Rep[Long]
```

```
def hashMatrixDeserializer(hash: Rep[HashStream[DenseMatrix[Double]]], k: Rep[String]): Rep
```

3.20 GrayscaleImage

<auto-generated stub>

3.20.1 Static methods

```
def apply(x: Rep[DenseMatrix[Double]]): Rep[GrayscaleImage]
```

3.20.2 Infix methods

```
def convolve(kernel: Rep[DenseMatrix[Double]]): Rep[GrayscaleImage]
```

```
def data(): Rep[DenseMatrix[Double]]
```

```
def downsample(rowFactor: Rep[Int], colFactor: Rep[Int])(sample: (Rep[GrayscaleImage]) => R
```

```
def histogram(): Rep[DenseVector[Int]]
```

```
def numCols(): Rep[Int]
```

```
def numRows(): Rep[Int]
```

```
def windowedFilter(rowDim: Rep[Int], colDim: Rep[Int])(block: (Rep[GrayscaleImage]) => Rep
```

3.21 HashStream

<auto-generated stub>

3.21.1 Static methods

```
def apply(table: Rep[String], deserialize: (Rep[HashStream[V]],Rep[String]) => Rep[V]): Rep
```

3.21.2 Infix methods

```
def apply(y: Rep[String]): Rep[V]
```

```
def close(): Rep[Unit]
```

```
def contains(y: Rep[String]): Rep[Boolean]
```

```
def get(y: Rep[String]): Rep[ForgeArray[Byte]]
```

```
def getAll(y: Rep[String]): Rep[ForgeArray[ForgeArray[Byte]]]
```

```
def keys(): Rep[ForgeArray[String]]
```

```
def mapValues(outFile: Rep[String], outDelim: Rep[String] = unit('' ''))(func: (Rep[Str
```

```
def open(): Rep[Unit]
```

```
def put(y: Rep[String], z: Rep[ForgeArray[Byte]]): Rep[Unit]
```

```
def putAll(y: Rep[ForgeArray[String]], z: Rep[ForgeArray[String]], v: Rep[ForgeArray[Forge
```

3.22 IndexVector

<auto-generated stub>

3.22.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Int]): Rep[IndexVector]
```

```
def apply(x: Rep[Int], y: Rep[Int], z: Rep[Boolean]): Rep[IndexVector]
```

```
def apply(x: Rep[DenseVector[Int]]): Rep[IndexVector]
```

```
def apply(x: Rep[DenseVector[Int]], y: Rep[Boolean]): Rep[IndexVector]
```

```
def apply(x: Rep[ForgeArray[Int]], isRow: Rep[Boolean] = true): Rep[IndexVector]
```

3.22.2 Infix methods

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[SparseVector[Int]]): Rep[SparseVector[Int]]
```

```
def *(y: Rep[SparseVectorView[Int]]): Rep[SparseVector[Int]]
```

```
def *(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseVector[Int]]
```

```
def **(y: Rep[DenseVector[Int]]): Rep[DenseMatrix[Int]]
```

```
def **(y: Rep[DenseVectorView[Int]]): Rep[DenseMatrix[Int]]
```

```
def **(y: Rep[SparseVector[Int]]): Rep[DenseMatrix[Int]]
```

```
def **(y: Rep[SparseVectorView[Int]]): Rep[DenseMatrix[Int]]
```

```
def *:*(y: Rep[DenseVector[Int]]): Rep[Int]
```

```
def *:*(y: Rep[DenseVectorView[Int]]): Rep[Int]
```

```
def *:*(y: Rep[SparseVector[Int]]): Rep[Int]
```

```
def *:*(y: Rep[SparseVectorView[Int]]): Rep[Int]
```

```
 def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
Math
```

```
def +(y: Rep[DenseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[SparseVector[Int]]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[SparseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[DenseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[SparseVector[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[SparseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[DenseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[SparseVector[Int]]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[SparseVectorView[Int]]): Rep[DenseVector[Int]]
```

```
def /(y: Rep[Int]): Rep[DenseVector[Int]]
```

```
def Clone(): Rep[IndexVector]
```

```
def abs(): Rep[DenseVector[Int]]
```

```
def apply(y: Rep[Int]): Rep[Int]
```

```
def apply(y: Rep[IndexVector]): Rep[IndexVector]
```

```
def apply(y: (Rep[Int]) => Rep[T]): Rep[DenseVector[T]]
```

```
def contains(y: Rep[Int]): Rep[Boolean]
```

```
def count(y: (Rep[Int]) => Rep[Boolean]): Rep[Int]
```

```
def distinct(): Rep[DenseVector[Int]]
```

```
def drop(y: Rep[Int]): Rep[IndexVector]
```

```
def exists(y: (Rep[Int]) => Rep[Boolean]): Rep[Boolean]
```

```
def exp(): Rep[DenseVector[Int]]
```

```
def filter(y: (Rep[Int]) => Rep[Boolean]): Rep[IndexVector]
```

```
def find(y: (Rep[Int]) => Rep[Boolean]): Rep[IndexVector]
```

```
def first(): Rep[Int]
```

```
def flatMap(y: (Rep[Int]) => Rep[DenseVector[R]]): Rep[DenseVector[R]]
```

```
def forall(y: (Rep[Int]) => Rep[Boolean]): Rep[Boolean]
```

```
def foreach(y: (Rep[Int]) => Rep[Unit]): Rep[Unit]
```

```
def histogram(): Rep[ForgeHashMap[Int,:doc:int]]
```

```
 def indices(): Rep[IndexVector]
```

Accessors

```
def intersect(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def isEmpty(): Rep[Boolean]
```

```
def isRow(): Rep[Boolean]
```

```
def last(): Rep[Int]
```

```
def length(): Rep[Int]
```

```
def log(): Rep[DenseVector[Int]]
```

```
def makeStrWithDelim(delim: Rep[String]): Rep[String]
```

```
def makeString(): Rep[String]
```

```
 def map(y: (Rep[Int]) => Rep[R]): Rep[DenseVector[R]]
Bulk
```

```
def max(): Rep[Int]
```

```
def maxIndex(): Rep[Int]
```

```
def mean()(implicit ev0: (Rep[Int]) => Rep[Double]): Rep[Double]
```

```
 def min(): Rep[Int]
Ordering
```

```
def minIndex(): Rep[Int]
```

```
def mutable(): Rep[DenseVector[Int]]
```

```
def partition(pred: (Rep[Int]) => Rep[Boolean]): Rep[Tup2[IndexVector,:doc:indexvector]]
```

```
def pprint(): Rep[Unit]
```

```
def prefixSum(): Rep[DenseVector[Int]]
```

```
def prod(): Rep[Int]
```

```
def reduce(y: (Rep[Int],Rep[Int]) => Rep[Int]): Rep[Int]
```

```
def replicate(y: Rep[Int], z: Rep[Int]): Rep[DenseMatrix[Int]]
```

```
def scanLeft(zero: Rep[R])(z: (Rep[R],Rep[Int]) => Rep[R]): Rep[DenseVector[R]]
```

```
def scanRight(zero: Rep[R])(z: (Rep[Int],Rep[R]) => Rep[R]): Rep[DenseVector[R]]
```

```
def slice(start: Rep[Int], end: Rep[Int]): Rep[IndexVector]
```

```
def stddev()(implicit ev0: (Rep[Int]) => Rep[Double]): Rep[Double]
```

```
def sum(): Rep[Int]
```

```
def t(): Rep[IndexVector]
```

```
def take(y: Rep[Int]): Rep[IndexVector]
```

```
 def toArray(): Rep[ForgeArray[Int]]
```

Data exchange

```
 def toBoolean()(implicit ev0: (Rep[Int]) => Rep[Boolean]): Rep[DenseVector[Boolean]]
```

Conversions

```
def toDense(): Rep[DenseVector[Int]]
```

```
def toDouble()(implicit ev0: (Rep[Int]) => Rep[Double]): Rep[DenseVector[Double]]
```

```
def toFloat()(implicit ev0: (Rep[Int]) => Rep[Float]): Rep[DenseVector[Float]]
```

```
def toInt()(implicit ev0: (Rep[Int]) => Rep[Int]): Rep[DenseVector[Int]]
```

```
def toString(): Rep[String]
```

```
def variance()(implicit ev0: (Rep[Int]) => Rep[Double]): Rep[Double]
```

```
def zip(y: Rep[DenseVector[B]])(z: (Rep[Int],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

```
def zip(y: Rep[DenseVectorView[B]])(z: (Rep[Int],Rep[B]) => Rep[R]): Rep[DenseVector[R]]
```

3.22.3 Implicit methods

```
def chainIndexToDenseIntOps(self: Rep[IndexVector]): DenseVectorDenseVectorIntOpsCls
```

```
def chainIndexToDenseOps(self: Rep[IndexVector]): DenseVectorDenseVectorOpsCls[Int]
```

```
def indexToDense(self: Rep[IndexVector]): Rep[DenseVector[Int]]
```

3.22.4 Related methods

```
def __equal(self: Rep[IndexVector], y: Rep[IndexVector]): Rep[Boolean]
```

```
def __equal(self: Rep[IndexVector], y: Rep[DenseVector[Int]]): Rep[Boolean]
```

```
def flatten(inds: Tuple2[Rep[Int],Rep[Int]], dims: Tuple2[Rep[Int],Rep[Int]]): Rep[Int]
```

```
def flatten(inds: Tuple3[Rep[Int],Rep[Int],Rep[Int]], dims: Tuple3[Rep[Int],Rep[Int],Rep[I
```

```
def flatten(inds: Tuple4[Rep[Int],Rep[Int],Rep[Int],Rep[Int]], dims: Tuple4[Rep[Int],Rep[I
```

```
def flatten(inds: Tuple5[Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[Int]], dims: Tuple5[Rep[I
```

```
def flatten(inds: Tuple6[Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[Int]], dims: Tupl
```

```
def indexvector_fromarray(x: Rep[ForgeArray[Int]], y: Rep[Boolean]): Rep[IndexVector]
```

```
def unflatten(i: Rep[Int], dims: Tuple2[Rep[Int],Rep[Int]]): Tuple2[Rep[Int],Rep[Int]]
```

```
def unflatten(i: Rep[Int], dims: Tuple3[Rep[Int],Rep[Int],Rep[Int]]): Tuple3[Rep[Int],Rep[I
```

```
def unflatten(i: Rep[Int], dims: Tuple4[Rep[Int],Rep[Int],Rep[Int],Rep[Int]]): Tuple4[Rep[I
```

```
def unflatten(i: Rep[Int], dims: Tuple5[Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[Int]]): Tup
```

```
def unflatten(i: Rep[Int], dims: Tuple6[Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[Int],Rep[I
```

3.23 KeyValueStore

<auto-generated stub>

3.23.1 Static methods

```
def apply(table: Rep[String], deserialize: (Rep[KeyValueStore[V]],Rep[String]) => Rep[V]):
```

3.23.2 Infix methods

```
def apply(y: Rep[String]): Rep[V]
```

```
def close(): Rep[Unit]
```

```
def contains(y: Rep[String]): Rep[Boolean]
```

```
def get(y: Rep[String]): Rep[ForgeArray[Byte]]
```

```
def getAll(y: Rep[String]): Rep[ForgeArray[ForgeArray[Byte]]]
```

```
def keys(): Rep[ForgeArray[String]]
```

```
def open(): Rep[Unit]
```

```
def put(y: Rep[String], z: Rep[ForgeArray[Byte]]): Rep[Unit]
```

```
def putAll(y: Rep[ForgeArray[String]], z: Rep[ForgeArray[String]], v: Rep[ForgeArray[Forge
```

3.24 RandomForest

<auto-generated stub>

3.24.1 Infix methods

```
def predict(testPt: Rep[DenseVector[Double]]): Rep[Tup2[Double,:doc:double]]
```

```
def probabilities(testPt: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def score(testPt: Rep[DenseVector[Double]]): Rep[Double]
```

```
def trees(): Rep[DenseVector[DecisionTree]]
```

3.24.2 Related methods

```
def alloc_forest(trees: Rep[DenseVector[DecisionTree]]): Rep[RandomForest]
```

3.25 Replicated

<auto-generated stub>

3.25.1 Static methods

```
def apply(x: Rep[ForgeArray[T]]): Rep[Replicated[T]]
```

3.25.2 Infix methods

```
def local(): Rep[T]
```

3.25.3 Implicit methods

```
def readLocal(r: Rep[Replicated[T]]): Rep[T]
```

3.26 SparseMatrix

<auto-generated stub>

3.26.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Int]): Rep[SparseMatrixBuildable[T]]
```

```
def diag(x: Rep[Int], y: Rep[SparseVector[T]]): Rep[SparseMatrix[T]]
```

```
def fromElements(numRows: Rep[Int], numCols: Rep[Int], nzElements: Rep[DenseVector[T]], nzI
```

```
def identity(x: Rep[Int], y: Rep[Int]): Rep[SparseMatrix[Double]]
```

```
def identity(x: Rep[Int]): Rep[SparseMatrix[Double]]
```

```
def rand(x: Rep[Int], y: Rep[Int], z: Rep[Double]): Rep[SparseMatrix[Double]]
```

```
def randf(x: Rep[Int], y: Rep[Int], z: Rep[Double]): Rep[SparseMatrix[Float]]
```

```
def randn(x: Rep[Int], y: Rep[Int], z: Rep[Double]): Rep[SparseMatrix[Double]]
```

```
def randnf(x: Rep[Int], y: Rep[Int], z: Rep[Double]): Rep[SparseMatrix[Float]]
```

```
def zeros(x: Rep[Int], y: Rep[Int]): Rep[SparseMatrix[Double]]
```

```
def zerosf(x: Rep[Int], y: Rep[Int]): Rep[SparseMatrix[Float]]
```

3.26.2 Infix methods

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def *:*(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def +(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def +(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def -(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def /(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def Clone(): Rep[SparseMatrix[T]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[T]
```

```
def apply(y: Rep[Int]): Rep[SparseVectorView[T]]
```

```
def apply(y: Rep[IndexVector]): Rep[SparseMatrix[T]]
```

```
def apply(y: Rep[IndexVector], z: IndexWildcard): Rep[SparseMatrix[T]]
```

```
def apply(rows: Rep[IndexVector], cols: Rep[IndexVector]): Rep[SparseMatrix[T]]
```

```
def apply(y: IndexWildcard, z: Rep[IndexVector]): Rep[SparseMatrix[T]]
```

```
def colIndices(): Rep[IndexVector]
```

```
def countnz(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def filterCols(y: (Rep[SparseVectorView[T]]) => Rep[Boolean]): Rep[SparseMatrix[T]]
```

```
def filterRows(y: (Rep[SparseVectorView[T]]) => Rep[Boolean]): Rep[SparseMatrix[T]]
```

```
def findCols(y: (Rep[SparseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
def findRows(y: (Rep[SparseVectorView[T]]) => Rep[Boolean]): Rep[IndexVector]
```

```
def foreachCol(y: (Rep[SparseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def foreachRow(y: (Rep[SparseVectorView[T]]) => Rep[Unit]): Rep[Unit]
```

```
def foreachnz(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def getCol(y: Rep[Int]): Rep[SparseVectorView[T]]
```

```
def getCols(y: Rep[IndexVector]): Rep[SparseMatrix[T]]
```

```
def getRow(y: Rep[Int]): Rep[SparseVectorView[T]]
```

```
def getRows(y: Rep[IndexVector]): Rep[SparseMatrix[T]]
```

```
def makeDimsStr(): Rep[String]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
def mapColsToVector(y: (Rep[SparseVectorView[T]]) => Rep[R]): Rep[SparseVector[R]]
```

```
def mapRowsToDenseVector(y: (Rep[SparseVectorView[T]]) => Rep[R]): Rep[DenseVector[R]]
```

```
def mapRowsToVector(y: (Rep[SparseVectorView[T]]) => Rep[R]): Rep[SparseVector[R]]
```

```
 def mapnz(y: (Rep[T]) => Rep[R]): Rep[SparseMatrix[R]]
Bulk
```

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def maxCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[SparseVector[T]]
```

```
def maxRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[SparseVector[T]]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
 def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
Ordering
```

```
def minCols()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[SparseVector[T]]
```

```
def minRows()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[SparseVector[T]]
```

```
def mutable(): Rep[SparseMatrixBuildable[T]]
```

```
def nnz(): Rep[Int]
```

```
def numCols(): Rep[Int]
```

```
 def numRows(): Rep[Int]
```

Accessors

```
def nz(asRow: Rep[Boolean] = true): Rep[DenseVector[T]]
```

```
def nzCols(): Rep[IndexVector]
```

```
def nzRows(): Rep[IndexVector]
```

```
def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

```
def rowIndices(): Rep[IndexVector]
```

```
def size(): Rep[Int]
```

```
def slice(startRow: Rep[Int], endRow: Rep[Int], startCol: Rep[Int], endCol: Rep[Int]): Rep
```

```
def sliceCols(start: Rep[Int], end: Rep[Int]): Rep[SparseMatrix[T]]
```

```
def sliceRows(start: Rep[Int], end: Rep[Int]): Rep[SparseMatrix[T]]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
def sumCols()(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def sumRows()(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
 def t(): Rep[SparseMatrix[T]]
```

Miscellaneous

```
def toDense(): Rep[DenseMatrix[T]]
```

```
def toString(): Rep[String]
```

```
def zipnz(y: Rep[DenseVector[B]])(z: (Rep[T],Rep[B]) => Rep[R]): Rep[SparseMatrix[R]]
```

3.26.3 Related methods

```
def __equal(self: Rep[SparseMatrix\[T\]], y: Rep[SparseMatrix[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[SparseMatrix\[T\]], y: Rep[DenseMatrix[T]]): Rep[Boolean]
```

3.27 SparseMatrixBuildable

<auto-generated stub>

3.27.1 Infix methods

```
def <<=(y: Rep[SparseVector[T]]): Rep[Unit]
```

```
def <<|=(y: Rep[SparseVector[T]]): Rep[Unit]
```

```
def append(i: Rep[Int], j: Rep[Int], y: Rep[T], alwaysWrite: Rep[Boolean] = true): Rep[Unit
```

```
def apply(i: Rep[Int], j: Rep[Int]): Rep[T]
```

```
 def finish(): Rep[SparseMatrix[T]]
```

Conversion to CSR

```
def insertCol(pos: Rep[Int], y: Rep[SparseVector[T]]): Rep[Unit]
```

```
def insertRow(pos: Rep[Int], y: Rep[SparseVector[T]]): Rep[Unit]
```

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
def mutable(): Rep[SparseMatrixBuildable[T]]
```

```
def nnz(): Rep[Int]
```

```
def numCols(): Rep[Int]
```

```
 def numRows(): Rep[Int]
```

Accessors

```
 def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

Miscellaneous

```
def removeCol(pos: Rep[Int]): Rep[Unit]
```

```
def removeCols(pos: Rep[Int], num: Rep[Int]): Rep[Unit]
```

```
def removeRow(pos: Rep[Int]): Rep[Unit]
```

```
def removeRows(pos: Rep[Int], num: Rep[Int]): Rep[Unit]
```

```
def size(): Rep[Int]
```

```
def toString(): Rep[String]
```

```
def update(y: Rep[Int], z: Rep[Int], v: Rep[T]): Rep[Unit]
```

3.28 SparseTrainingSet

<auto-generated stub>

3.28.1 Static methods

```
def apply(x: Rep[SparseMatrix[D]], y: Rep[DenseVector[L]]): Rep[SparseTrainingSet[D,L]]
```

3.28.2 Infix methods

```
def apply(y: Rep[Int], z: Rep[Int]): Rep[D]
```

```
def apply(y: Rep[Int]): Rep[SparseVectorView[D]]
```

```
def data(): Rep[SparseMatrix[D]]
```

```
def labels(): Rep[DenseVector[L]]
```

```
def numFeatures(): Rep[Int]
```

```
def numSamples(): Rep[Int]
```

3.29 SparseVector

<auto-generated stub>

3.29.1 Static methods

```
def apply(x: Rep[Int], y: Rep[Boolean]): Rep[SparseVector[T]]
```

```
def fromElements(length: Rep[Int], isRow: Rep[Boolean], nzIndices: Rep[IndexVector], nzEle
```

```
def fromFunc(x: Rep[Int], y: Rep[Boolean], z: Rep[IndexVector], v: (Rep[Int]) => Rep[T]): R
```

```
def fromSortedElements(length: Rep[Int], isRow: Rep[Boolean], nzIndices: Rep[IndexVector],
```

```
def rand(length: Rep[Int], sparsity: Rep[Double]): Rep[SparseVector[Double]]
```

```
def randf(length: Rep[Int], sparsity: Rep[Double]): Rep[SparseVector[Float]]
```

```
def zeros(x: Rep[Int]): Rep[SparseVector[Double]]
```

```
def zerosf(x: Rep[Int]): Rep[SparseVector[Float]]
```

3.29.2 Infix methods

```
def *(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def **(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def *:*(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def +(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def +(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def +(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def -(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def -(y: Rep[T])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def /(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def /(y: Rep[T])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def <<(y: Rep[T]): Rep[SparseVector[T]]
```

```
def <<(y: Rep[SparseVector[T]]): Rep[SparseVector[T]]
```

```
def <<=(y: Rep[T]): Rep[Unit]
```

```
def <<=(y: Rep[SparseVector[T]]): Rep[Unit]
```

```
def Clone(): Rep[SparseVector[T]]
```

```
def abs()(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def apply(y: Rep[Int]): Rep[T]
```

```
def apply(y: Rep[IndexVector]): Rep[SparseVector[T]]
```

```
def clear(): Rep[Unit]
```

```
def contains(y: Rep[T]): Rep[Boolean]
```

```
def copyFrom(pos: Rep[Int], xs: Rep[SparseVector[T]]): Rep[Unit]
```

```
def countnz(y: (Rep[T]) => Rep[Boolean]): Rep[Int]
```

```
def distinct(): Rep[DenseVector[T]]
```

```
def drop(y: Rep[Int]): Rep[SparseVector[T]]
```

```
def filternz(y: (Rep[T]) => Rep[Boolean]): Rep[SparseVector[T]]
```

```
def findnz(y: (Rep[T]) => Rep[Boolean]): Rep[IndexVector]
```

```
def first(): Rep[T]
```

```
def firstnz(): Rep[T]
```

```
def foreachnz(y: (Rep[T]) => Rep[Unit]): Rep[Unit]
```

```
def indices(): Rep[IndexVector]
```

```
def insert(y: Rep[Int], z: Rep[T]): Rep[Unit]
```

```
def insertAll(pos: Rep[Int], xs: Rep[SparseVector[T]]): Rep[Unit]
```

```
def isEmpty(): Rep[Boolean]
```

```
def isRow(): Rep[Boolean]
```

```
def last(): Rep[T]
```

```
def lastnz(): Rep[T]
```

```
 def length(): Rep[Int]
```

Accessors

```
def makeString()(implicit ev0: Stringable[T]): Rep[String]
```

```
 def mapnz(y: (Rep[T]) => Rep[R]): Rep[SparseVector[R]]
```

Bulk

```
def max()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def mean()(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
 def min()(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

Ordering

```
def mt(): Rep[Unit]
```

```
def mutable(): Rep[SparseVector[T]]
```

```
def nnz(): Rep[Int]
```

```
def nz(): Rep[DenseVectorView[T]]
```

```
def pprint()(implicit ev0: Stringable[T]): Rep[Unit]
```

```
def reducenz(y: (Rep[T],Rep[T]) => Rep[T])(implicit ev0: Arith[T]): Rep[T]
```

```
def remove(y: Rep[Int]): Rep[Unit]
```

```
def removeAll(pos: Rep[Int], len: Rep[Int]): Rep[Unit]
```

```
def slice(start: Rep[Int], end: Rep[Int]): Rep[SparseVector[T]]
```

```
def sum()(implicit ev0: Arith[T]): Rep[T]
```

```
 def t(): Rep[SparseVector[T]]
```

Miscellaneous

```
def take(y: Rep[Int]): Rep[SparseVector[T]]
```

```
def toDense(): Rep[DenseVector[T]]
```

```
def toString(): Rep[String]
```

```
def trim(): Rep[Unit]
```

```
def update(pos: Rep[Int], e: Rep[T]): Rep[Unit]
```

```
def update(indices: Rep[IndexVector], e: Rep[T]): Rep[Unit]
```

```
def update(indices: Rep[IndexVector], v: Rep[SparseVector[T]]): Rep[Unit]
```

3.29.3 Implicit methods

```
def dist(x: Rep[SparseVector[Double]], y: Rep[SparseVector[Double]]): Rep[Double]
```

3.29.4 Related methods

```
def __equal(self: Rep[SparseVector\[T\]], y: Rep[DenseVector[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[SparseVector\[T\]], y: Rep[SparseVectorView[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[SparseVector\[T\]], y: Rep[SparseVector[T]]): Rep[Boolean]
```

```
def dist(x: Rep[SparseVector[Double]], y: Rep[SparseVector[Double]], z: DistanceMetric): Re
```

3.30 SparseVectorView

<auto-generated stub>

3.30.1 Static methods

```
def apply(x: Rep[SparseMatrix[T]], y: Rep[Long], z: Rep[Int], v: Rep[Int], w: Rep[Boolean])
```

3.30.2 Infix methods

```
def *(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[T])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def *(y: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def *:*(y: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def *:*(y: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def Clone(): Rep[SparseVector[T]]
```

```
def apply(y: Rep[Int]): Rep[T]
```

```
def indices(): Rep[IndexVector]
```

```
def isRow(): Rep[Boolean]
```

```
def length(): Rep[Int]
```

```
def nnz(): Rep[Int]
```

```
def nz(): Rep[DenseVectorView[T]]
```

```
def toDense(): Rep[DenseVector[T]]
```

```
def toSparse(): Rep[SparseVector[T]]
```

```
def toString(): Rep[String]
```

3.30.3 Implicit methods

```
def chainViewToSparseOps(self: Rep[SparseVectorView\[T\]]): SparseVectorSparseVectorOpsCls
```

```
def viewToSparse(self: Rep[SparseVectorView\[T\]]): Rep[SparseVector[T]]
```

3.30.4 Related methods

```
def __equal(self: Rep[SparseVectorView\[T\]], y: Rep[SparseVectorView[T]]): Rep[Boolean]
```

```
def __equal(self: Rep[SparseVectorView\[T\]], y: Rep[SparseVector[T]]): Rep[Boolean]
```

3.31 Tup2

<auto-generated stub>

3.31.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def toString(): Rep[String]
```

3.31.2 Related methods

```
def pack(t: Tuple2[Rep[A],Rep[B]]): Rep[Tup2[A,B]]
```

```
def pack(x: Tuple2[Var[A],Rep[B]]): Rep[Tup2[A,B]]
```

```
def pack(x: Tuple2[Rep[A],Var[B]]): Rep[Tup2[A,B]]
```

```
def pack(x: Tuple2[Var[A],Var[B]]): Rep[Tup2[A,B]]
```

```
def unpack(t: Rep[Tup2[A,B]]): Tuple2[Rep[A],Rep[B]]
```

3.32 Tup3

<auto-generated stub>

3.32.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def toString(): Rep[String]
```

3.32.2 Related methods

```
def pack(t: Tuple3[Rep[A],Rep[B],Rep[C]]): Rep[Tup3[A,B,C]]
```

```
def unpack(t: Rep[Tup3[A,B,C]]): Tuple3[Rep[A],Rep[B],Rep[C]]
```

3.33 Tup4

<auto-generated stub>

3.33.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def toString(): Rep[String]
```

3.33.2 Related methods

```
def pack(t: Tuple4[Rep[A],Rep[B],Rep[C],Rep[D]]): Rep[Tup4[A,B,C,D]]
```

```
def unpack(t: Rep[Tup4[A,B,C,D]]): Tuple4[Rep[A],Rep[B],Rep[C],Rep[D]]
```

3.34 Tup5

<auto-generated stub>

3.34.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def _5(): Rep[E]
```

```
def toString(): Rep[String]
```

3.34.2 Related methods

```
def pack(t: Tuple5[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E]]): Rep[Tup5[A,B,C,D,E]]
```

```
def unpack(t: Rep[Tup5[A,B,C,D,E]]): Tuple5[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E]]
```

3.35 Tup6

<auto-generated stub>

3.35.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def _5(): Rep[E]
```

```
def _6(): Rep[F]
```

```
def toString(): Rep[String]
```

3.35.2 Related methods

```
def pack(t: Tuple6[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F]]): Rep[Tup6[A,B,C,D,E,F]]
```

```
def unpack(t: Rep[Tup6[A,B,C,D,E,F]]): Tuple6[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F]]
```

3.36 Tup7

<auto-generated stub>

3.36.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def _5(): Rep[E]
```

```
def _6(): Rep[F]
```

```
def _7(): Rep[G]
```

```
def toString(): Rep[String]
```

3.36.2 Related methods

```
def pack(t: Tuple7[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F],Rep[G]]): Rep[Tup7[A,B,C,D,E,
```

```
def unpack(t: Rep[Tup7[A,B,C,D,E,F,G]]): Tuple7[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F],
```

3.37 Tup8

<auto-generated stub>

3.37.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def _5(): Rep[E]
```

```
def _6(): Rep[F]
```

```
def _7(): Rep[G]
```

```
def _8(): Rep[H]
```

```
def toString(): Rep[String]
```

3.37.2 Related methods

```
def pack(t: Tuple8[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F],Rep[G],Rep[H]]): Rep[Tup8[A,B,
```

```
def unpack(t: Rep[Tup8[A,B,C,D,E,F,G,H]]): Tuple8[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F]
```

3.38 Tup9

<auto-generated stub>

3.38.1 Infix methods

```
def _1(): Rep[A]
```

```
def _2(): Rep[B]
```

```
def _3(): Rep[C]
```

```
def _4(): Rep[D]
```

```
def _5(): Rep[E]
```

```
def _6(): Rep[F]
```

```
def _7(): Rep[G]
```

```
def _8(): Rep[H]
```

```
def _9(): Rep[I]
```

```
def toString(): Rep[String]
```

3.38.2 Related methods

```
def pack(t: Tuple9[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep[F],Rep[G],Rep[H],Rep[I]]): Rep[Tu
```

```
def unpack(t: Rep[Tup9[A,B,C,D,E,F,G,H,I]]): Tuple9[Rep[A],Rep[B],Rep[C],Rep[D],Rep[E],Rep
```

3.39 UTriangle

<auto-generated stub>

3.39.1 Infix methods

```
def N(): Rep[Int]
```

```
def apply(n: Rep[Int]): Rep[Tup2[Int,:doc:int]]
```

```
def contains(i: Rep[Int], j: Rep[Int]): Rep[Boolean]
```

```
def includeDiagonal(): Rep[Boolean]
```

```
def size(): Rep[Int]
```

3.39.2 Related methods

```
def utriangle(N: Rep[Int], includeDiagonal: Rep[Boolean] = true): Rep[UTriangle]
```

Objects

4.1 DateFeature

<auto-generated stub>

4.1.1 Static methods

```
def apply(x: Rep[String]): Rep[org.joda.time.format.DateTimeFormatter]
```

```
def day(x: Rep[Double]): Rep[Int]
```

```
def daysBetween(x: Rep[Double], y: Rep[Double]): Rep[Int]
```

```
def hour(x: Rep[Double]): Rep[Int]
```

```
def month(x: Rep[Double]): Rep[Int]
```

```
def monthsBetween(x: Rep[Double], y: Rep[Double]): Rep[Int]
```

```
def weekday(x: Rep[Double]): Rep[Int]
```

```
def year(x: Rep[Double]): Rep[Int]
```

```
def yearsBetween(x: Rep[Double], y: Rep[Double]): Rep[Int]
```

4.2 Math

<auto-generated stub>

4.2.1 Static methods

```
def abs(x: Rep[Double]): Rep[Double]
```

```
def acos(x: Rep[Double]): Rep[Double]
```

```
def asin(x: Rep[Double]): Rep[Double]
```

```
def atan(x: Rep[Double]): Rep[Double]
```

```
def atan2(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def bitcount(x: Rep[Long]): Rep[Int]
```

```
def ceil(x: Rep[Double]): Rep[Double]
```

```
def cos(x: Rep[Double]): Rep[Double]
```

```
def cosh(x: Rep[Double]): Rep[Double]
```

```
def exp(x: Rep[Double]): Rep[Double]
```

```
def floor(x: Rep[Double]): Rep[Double]
```

```
def log(x: Rep[Double]): Rep[Double]
```

```
def log10(x: Rep[Double]): Rep[Double]
```

```
def max(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def min(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def pow(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def round(x: Rep[Double]): Rep[Long]
```

```
def sin(x: Rep[Double]): Rep[Double]
```

```
def sinh(x: Rep[Double]): Rep[Double]
```

```
def sqrt(x: Rep[Double]): Rep[Double]
```

```
def tan(x: Rep[Double]): Rep[Double]
```

```
def tanh(x: Rep[Double]): Rep[Double]
```

4.2.2 Related methods

```
def E(): Rep[Double]
```

```
def INF(): Rep[Double]
```

```
def Pi(): Rep[Double]
```

```
def nINF(): Rep[Double]
```

CHAPTER 5

Primitives

5.1 Boolean

<auto-generated stub>

5.1.1 Infix methods

```
def &&(y: Rep[Boolean]): Rep[Boolean]
```

Boolean AND

```
def unary_!(): Rep[Boolean]
```

Boolean inverse

```
def ||(y: Rep[Boolean]): Rep[Boolean]
```

Boolean OR

5.2 Double

<auto-generated stub>

5.2.1 Infix methods

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def unary_-(): Rep[Double]
```

Negation

```
def ~^(y: Rep[Int]): Rep[Double]
```

5.3 Float

<auto-generated stub>

5.3.1 Infix methods

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def unary_-(): Rep[Float]
```

Negation

5.4 ForgeArray

<auto-generated stub>

5.4.1 Infix methods

```
def apply(y: Rep[Int]): Rep[T]
```

```
def length(): Rep[Int]
```

5.5 ForgeHashMap

<auto-generated stub>

5.5.1 Infix methods

```
def toVector(): Rep[DenseVector[R]]
```

5.6 Int

<auto-generated stub>

5.6.1 Infix methods

```
def %(y: Rep[Int]): Rep[Int]
```

```
def &(y: Rep[Int]): Rep[Int]
```

```
def *(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def *(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def *(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def *(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def *(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def *(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def +(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def +(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def +(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def +(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def +(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def +(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def -(y: Rep[DenseVector[Int]]): Rep[DenseVector[Int]]
```

```
def -(y: Rep[DenseVector[Float]]): Rep[DenseVector[Float]]
```

```
def -(y: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def -(y: Rep[DenseMatrix[Int]]): Rep[DenseMatrix[Int]]
```

```
def -(y: Rep[DenseMatrix[Float]]): Rep[DenseMatrix[Float]]
```

```
def -(y: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def ::(start: Rep[Int]): Rep[IndexVector]
```

```
def <<(y: Rep[Int]): Rep[Int]
```

```
def >>(y: Rep[Int]): Rep[Int]
```

```
def >>>(y: Rep[Int]): Rep[Int]
```

```
def unary_-(): Rep[Int]
```

Negation

```
def unary_~(): Rep[Int]
```

```
def |(y: Rep[Int]): Rep[Int]
```

5.7 Long

<auto-generated stub>

5.7.1 Infix methods

```
def %(y: Rep[Long]): Rep[Long]
```

```
def &(y: Rep[Long]): Rep[Long]
```

```
def <<(y: Rep[Int]): Rep[Long]
```

```
def >>(y: Rep[Int]): Rep[Long]
```

```
def >>>(y: Rep[Int]): Rep[Long]
```

```
def ^(y: Rep[Long]): Rep[Long]
```

```
def unary_-(): Rep[Long]
```

Negation

```
def unary_~(): Rep[Long]
```

```
def |(y: Rep[Long]): Rep[Long]
```

5.8 String

Exits the program with a fatal exception message. @param message

5.8.1 Infix methods

```
def contains(y: Rep[String]): Rep[Boolean]
```

```
def endsWith(y: Rep[String]): Rep[Boolean]
```

```
def fcharAt(y: Rep[Int]): Rep[Char]
```

```
def fsplit(y: Rep[String], numSplits: Rep[Int] = 0): Rep[ForgeArray[String]]
```

```
def getBytes(): Rep[ForgeArray[Byte]]
```

```
def length(): Rep[Int]
```

```
def replaceAllLiterally(y: Rep[String], z: Rep[String]): Rep[String]
```

```
def slice(y: Rep[Int], z: Rep[Int]): Rep[String]
```

```
def split(y: Rep[String], numSplits: Rep[Int] = 0): Rep[ForgeArray[String]]
```

```
def startsWith(y: Rep[String]): Rep[Boolean]
```

```
def substring(y: Rep[Int]): Rep[String]
```

```
def substring(y: Rep[Int], z: Rep[Int]): Rep[String]
```

```
def toBoolean(): Rep[Boolean]
```

```
def toDouble(): Rep[Double]
```

```
def toFloat(): Rep[Float]
```

```
def toInt(): Rep[Int]
```

```
def toLong(): Rep[Long]
```

```
def toLowerCase(): Rep[String]
```

```
def toUpperCase(): Rep[String]
```

```
def trim(): Rep[String]
```

5.9 java.io.DataInputStream

<auto-generated stub>

5.9.1 Infix methods

```
def available(): Rep[Int]
```

```
def fclose(): Rep[Unit]
```

```
def readBoolean(): Rep[Boolean]
```

```
def readDouble(): Rep[Double]
```

```
def readInt(): Rep[Int]
```

```
def readLong(): Rep[Long]
```

```
def readShort(): Rep[Short]
```

5.10 org.joda.time.format.DateTimeFormatter

<auto-generated stub>

5.10.1 Infix methods

```
def apply(y: Rep[String]): Rep[Double]
```

CHAPTER 6

Generic Methods

6.1 T

<auto-generated stub>

6.1.1 Infix methods

```
def +(y: String): Rep[String]
```

```
def AsInstanceOf(): Rep[R]
```

Casts this to the given type

```
def IsInstanceOf(): Rep[Boolean]
```

Checks if this is the specified type

```
def toDouble(): Rep[Double]
```

Converts this numeric value to an Double

```
def toFloat(): Rep[Float]
```

Converts this numeric value to a Float

```
def toInt(): Rep[Int]
```

Converts this numeric value to an Int

```
def toLong(): Rep[Long]
```

Converts this numeric value to a Long

```
def unsafeImmutable(): Rep[T]
```

Gives a hint to the compiler to consider this value as immutable for future operations Note that this operation is unsafe as it does not prevent mutations prior to this call. For example, in:

```
val x = Array[Int](32)
for (i <- 0 until N) {
  x(i) = i
  f(x.unsafeImmutable)
}
```

This will tell the compiler that x is not mutable in the function x, but x can still be mutated after this call has been made (in the next iteration of the loop, for example).

```
def unsafeMutable(): Rep[T]
```

Enables unsafe mutation of this value.

CHAPTER 7

Operations

7.1 BasicMath

<auto-generated stub>

7.1.1 Related methods

```
def abs(x: Rep[Double]): Rep[Double]
```

```
def abs(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def abs(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def abs(x: Rep[IndexVector]): Rep[DenseVector[Int]]
```

```
def abs(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def abs(x: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def abs(x: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def acos(x: Rep[Double]): Rep[Double]
```

```
def asin(x: Rep[Double]): Rep[Double]
```

```
def atan(x: Rep[Double]): Rep[Double]
```

```
def atan2(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def ceil(x: Rep[Double]): Rep[Int]
```

```
def cos(x: Rep[Double]): Rep[Double]
```

```
def cosh(x: Rep[Double]): Rep[Double]
```

```
def exp(x: Rep[Double]): Rep[Double]
```

```
def exp(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def exp(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def exp(x: Rep[IndexVector]): Rep[DenseVector[Int]]
```

```
def exp(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def floor(x: Rep[Double]): Rep[Int]
```

```
def log(x: Rep[Double]): Rep[Double]
```

```
def log(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def log(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def log(x: Rep[IndexVector]): Rep[DenseVector[Int]]
```

```
def log(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def log10(x: Rep[Double]): Rep[Double]
```

```
def max(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def max(x: Rep[DenseVector[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def max(x: Rep[DenseVectorView[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def max(x: Rep[IndexVector]): Rep[Int]
```

```
def max(x: Rep[DenseMatrix[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def max(x: Rep[SparseVector[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def max(x: Rep[SparseMatrix[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def max(x: Rep[T]*)(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def mean(x: Rep[DenseVector[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def mean(x: Rep[DenseVectorView[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def mean(x: Rep[IndexVector]): Rep[Double]
```

```
def mean(x: Rep[DenseMatrix[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def mean(x: Rep[SparseVector[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def mean(x: Rep[SparseMatrix[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def mean(x: Rep[T]*)(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def median(x: Rep[DenseVector[T]])(implicit ev0: Numeric[T],ev1: Ordering[T]): Rep[T]
```

```
def median(x: Rep[T]*)(implicit ev0: Numeric[T],ev1: Ordering[T]): Rep[T]
```

```
def min(x: Rep[T], y: Rep[T]): Rep[T]
```

```
def min(x: Rep[DenseVector[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def min(x: Rep[DenseVectorView[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def min(x: Rep[IndexVector]): Rep[Int]
```

```
def min(x: Rep[DenseMatrix[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def min(x: Rep[SparseVector[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def min(x: Rep[SparseMatrix[T]])(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def min(x: Rep[T]*)(implicit ev0: Ordering[T],ev1: HasMinMax[T]): Rep[T]
```

```
def normalize(x: Rep[DenseVector[Double]]): Rep[DenseVector[Double]]
```

```
def normalize(x: Rep[DenseVector[Double]], y: NormalizeMethod): Rep[DenseVector[Double]]
```

```
def normalize(x: Rep[DenseMatrix[Double]]): Rep[DenseMatrix[Double]]
```

```
def normalize(x: Rep[DenseMatrix[Double]], y: NormalizeMethod): Rep[DenseMatrix[Double]]
```

```
def normalizeStdScalarUsing(e: Rep[Double], avg: Rep[Double], stddev: Rep[Double]): Rep[Dou
```

```
def normalizeStdUsing(v: Rep[DenseVector[Double]], avg: Rep[Double], stddev: Rep[Double]):
```

```
def normalizeUnityScalarUsing(e: Rep[Double], minVal: Rep[Double], maxVal: Rep[Double]): Re
```

```
def normalizeUnityUsing(v: Rep[DenseVector[Double]], minVal: Rep[Double], maxVal: Rep[Doubl
```

```
def normpdf(x: Rep[Double], mu: Rep[Double], sigma: Rep[Double]): Rep[Double]
```

```
def normpdf(x: Rep[DenseVector[Double]], mu: Rep[DenseVector[Double]], sigma: Rep[DenseVect
```

```
def normpdf(x: Rep[DenseMatrix[Double]], mu: Rep[DenseMatrix[Double]], sigma: Rep[DenseMatr
```

```
def normpdf(x: Rep[DenseVector[Double]], mu: Rep[Double], sigma: Rep[DenseVector[Double]])
```

```
def normpdf(x: Rep[DenseVector[Double]], mu: Rep[DenseVector[Double]], sigma: Rep[Double])
```

```
def normpdf(x: Rep[DenseVector[Double]], mu: Rep[Double], sigma: Rep[Double]): Rep[DenseVec
```

```
def normpdf(x: Rep[DenseMatrix[Double]], mu: Rep[Double], sigma: Rep[DenseMatrix[Double]])
```

```
def normpdf(x: Rep[DenseMatrix[Double]], mu: Rep[DenseMatrix[Double]], sigma: Rep[Double])
```

```
def normpdf(x: Rep[DenseMatrix[Double]], mu: Rep[Double], sigma: Rep[Double]): Rep[DenseMat
```

```
def pow(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def prod(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def prod(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def prod(x: Rep[IndexVector]): Rep[Int]
```

```
def prod(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def round(x: Rep[Double]): Rep[Int]
```

```
def sigmoid(x: Rep[Double]): Rep[Double]
```

```
def sin(x: Rep[Double]): Rep[Double]
```

```
def sinh(x: Rep[Double]): Rep[Double]
```

```
def sqrt(x: Rep[Double]): Rep[Double]
```

```
def square(x: Rep[Double]): Rep[Double]
```

```
def square(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def square(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[DenseVector[T]]
```

```
def square(x: Rep[IndexVector]): Rep[DenseVector[Int]]
```

```
def square(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[DenseMatrix[T]]
```

```
def square(x: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[SparseVector[T]]
```

```
def square(x: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[SparseMatrix[T]]
```

```
def stddev(x: Rep[DenseVector[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def stddev(x: Rep[DenseVectorView[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def stddev(x: Rep[IndexVector]): Rep[Double]
```

```
def stddev(x: Rep[DenseMatrix[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def sum(x: Rep[DenseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def sum(x: Rep[DenseVectorView[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def sum(x: Rep[IndexVector]): Rep[Int]
```

```
def sum(x: Rep[DenseMatrix[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def sum(x: Rep[SparseVector[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def sum(x: Rep[SparseMatrix[T]])(implicit ev0: Arith[T]): Rep[T]
```

```
def tan(x: Rep[Double]): Rep[Double]
```

```
def tanh(x: Rep[Double]): Rep[Double]
```

```
def variance(x: Rep[DenseVector[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

```
def variance(x: Rep[DenseVectorView[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double
```

```
def variance(x: Rep[IndexVector]): Rep[Double]
```

```
def variance(x: Rep[DenseMatrix[T]])(implicit ev0: (Rep[T]) => Rep[Double]): Rep[Double]
```

7.2 Classifier

<auto-generated stub>

7.2.1 Related methods

```
def logreg(data: Rep[TS[Double,:doc:boolean]], stochastic: Rep[Boolean] = true, initLearni
```

Logistic regression with dense parameters. The training set can be dense or sparse.

```
def rforest(trainingSet: Rep[DenseTrainingSet[Double,:doc:boolean]], numTrees: Rep[Int] = 1
```

```
def sparseLogreg(data: Rep[SparseTrainingSet[Double,:doc:boolean]], stochastic: Rep[Boolean
```

Logistic regression with sparse parameters. The training set must be sparse. FIXME: With regularization turned on, this is not producing exactly the same results as dense logreg. However, it still seems to work, more or less.

7.3 Control

<auto-generated stub>

7.3.1 Related methods

```
def getNumSockets(): Rep[Int]
```

```
def getSocket(): Rep[Int]
```

```
def replicate(x: Rep[FactorGraph]): Rep[Replicated[FactorGraph]]
```

```
def untilconverged(x: Rep[T], tol: Rep[Double] = unit(.001), minIter: Rep[Int] = 1, maxIter
```

```
def untilconverged_buffered(x: Rep[T], tol: Rep[Double] = unit(.001), minIter: Rep[Int] = 1
```

```
def untilconverged_withdiff(x: Rep[T], tol: Rep[Double] = unit(.001), minIter: Rep[Int] = 1
```

7.4 FString

<auto-generated stub>

7.4.1 Related methods

```
def optila_fmt_str(x: Rep[T]): Rep[String]
```

7.5 FeatureHelper

<auto-generated stub>

7.5.1 Related methods

```
def dumpUniqueMappings(x: Rep[String]): Rep[Unit]
```

```
def getUniqueIds(): Rep[ForgeArray[Int]]
```

```
def getUniqueNames(): Rep[ForgeArray[String]]
```

```
def loadUniqueMappings(x: Rep[String]): Rep[Unit]
```

```
def reverseUnique(x: Rep[Int]): Rep[String]
```

```
def unique(x: Rep[String]): Rep[Int]
```

7.6 IO

<auto-generated stub>

7.6.1 Related methods

```
def readFactorGraph(factorsPath: Rep[String], variablesPath: Rep[String], weightsPath: Rep
```

7.7 LAio

<auto-generated stub>

7.7.1 Related methods

```
def deleteFile(x: Rep[String]): Rep[Unit]
```

```
def fileExists(x: Rep[String]): Rep[Boolean]
```

```
def fileLength(x: Rep[String]): Rep[Long]
```

```
def readMatrix(path: Rep[String]): Rep[DenseMatrix[Double]]
```

```
def readMatrix(path: Rep[String], delim: Rep[String]): Rep[DenseMatrix[Double]]
```

```
def readMatrix(path: Rep[String], schemaBldr: (Rep[String]) => Rep[Elem], delim: Rep[Strin
```

```
def readMatrixAndParse(path: Rep[String], schemaBldr: (Rep[DenseVector[String]]) => Rep[De
```

```
 def readVector(path: Rep[String]): Rep[DenseVector[Double]]
For fusion and cluster execution, reads should be pure. however, in that case we need a di
```

```
def readVector(path: Rep[String], schemaBldr: (Rep[String]) => Rep[Elem]): Rep[DenseVector
```

```
def readVectorAndParse(path: Rep[String], schemaBldr: (Rep[DenseVector[String]]) => Rep[Ele
```

```
def writeMatrix(m: Rep[DenseMatrix[Elem]], path: Rep[String], delim: Rep[String] = unit('' 
```

```
def writeVector(v: Rep[DenseVector[Elem]], path: Rep[String]): Rep[Unit]
```

7.8 LinAlg

<auto-generated stub>

7.8.1 Related methods

```
def chol(A: Rep[DenseMatrix[Double]], tri: Rep[String] = unit(``upper'')): Rep[DenseMatrix
```

```
def det(x: Rep[DenseMatrix[T]]): Rep[T]
```

```
def lu(x: Rep[DenseMatrix[Double]]): Tuple3[Rep[DenseMatrix[Double]],Rep[DenseMatrix[Double
```

7.9 MLio

<auto-generated stub>

7.9.1 Related methods

```
def readARFF(path: Rep[String], schemaBldr: (Rep[DenseVector[String]]) => Rep[Row]): Rep[De
```

7.10 Misc

Miscellaneous operations, including assertions, console printing, and exception throwing.

7.10.1 Related methods

```
def __ifThenElse(x: Rep[Boolean], y:  => Rep[T], z:  => Rep[T]): Rep[T]
```

```
def __whileDo(x:  => Rep[Boolean], y:  => Rep[Unit]): Rep[Unit]
```

```
def exit(x: Rep[Int]): Rep[Nothing]
```

```
def fassert(cond: Rep[Boolean], message: Rep[String]): Rep[Unit]
```

If the condition is true, does nothing. If it is false, terminates the program with the specified exception message. Assertions are only evaluated in the library backend and during compiler debugging. Otherwise, they are ignored to enable more fusion and code motion opportunities.

- **cond** - condition to be checked
- **message** - exception message to be shown to the user if the assertion fails

---

```
def fatal(x: Rep[String]): Rep[Nothing]
```

```
def getMaxHeapSize(): Rep[Long]
```

Returns the maximum heap size from the JVM

```
def print(x: Rep[Any]): Rep[Unit]
```

```
def println(value: Rep[Any]): Rep[Unit]
```

Prints the specified value to the console *with* a newline. At runtime, this will attempt to use the toString method to cast the value to string if one exists

---

```
def println(): Rep[Unit]
```

Prints a newline to the console

7.11 Primitive

<auto-generated stub>

7.11.1 Implicit methods

```
def dist(x: Rep[Int], y: Rep[Int]): Rep[Double]
```

```
def dist(x: Rep[Double], y: Rep[Double]): Rep[Double]
```

```
def repFloat2ToRepDouble(x: Rep[Float]): Rep[Double]
```

Enables implicit conversion from Float to Double

```
def repInt2ToRepDouble(x: Rep[Int]): Rep[Double]
```

Enables implicit conversion from Int to Double

```
def repInt2ToRepFloat(x: Rep[Int]): Rep[Float]
```

Enables implicit conversion from Int to Float

```
def repInt2ToRepLong(x: Rep[Int]): Rep[Long]
```

Enables implicit conversion from Int to Long

7.11.2 Related methods

```
def numeric_to_double(x: Rep[T]): Rep[Double]
```

```
def numeric_to_float(x: Rep[T]): Rep[Float]
```

```
def numeric_to_int(x: Rep[T]): Rep[Int]
```

```
def numeric_to_long(x: Rep[T]): Rep[Long]
```

7.12 Rand

<auto-generated stub>

7.12.1 Related methods

```
def random(): Rep[A]
```

```
def randomElem(x: Rep[DenseVector[A]]): Rep[A]
```

```
def randomGaussian(): Rep[Double]
```

```
def randomInt(x: Rep[Int]): Rep[Int]
```

```
def reseed(): Rep[Unit]
```

```
def sample(v: Rep[IndexVector], pct: Rep[Double]): Rep[IndexVector]
```

```
def sample(v: Rep[DenseVector[A]], pct: Rep[Double]): Rep[DenseVector[A]]
```

```
def sample(m: Rep[DenseMatrix[A]], pct: Rep[Double], sampleRows: Rep[Boolean] = true): Rep
```

```
def shuffle(x: Rep[IndexVector]): Rep[IndexVector]
```

```
def shuffle(x: Rep[DenseVector[A]]): Rep[DenseVector[A]]
```

```
def shuffle(x: Rep[DenseMatrix[A]]): Rep[DenseMatrix[A]]
```

7.13 Validate

<auto-generated stub>

7.13.1 Related methods

```
def AUC(unsortedROCs: Rep[DenseVector[Tup2[Double,:doc:double]]]): Rep[Double]
```

```
def ROC(x: Rep[DenseMatrix[Int]]): Rep[Tup2[Double,:doc:double]]
```

```
def ROCCurve(testSet: Rep[TS[T,:doc:boolean]], classify: (Rep[Double]) => (Rep[Int]) => Rep
```

```
def ROCCurveBatch(testSet: Rep[TS[T,:doc:boolean]], classify: (Rep[Double]) => (Rep[IndexVe
```

```
def accuracy(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
 def confusionMatrix(testSet: Rep[TS[T,:doc:boolean]], classify: (Rep[Int]) => Rep[Boolean]
```

Generate a confusion matrix for the given classifier and testSet.

```
 def confusionMatrixBatch(testSet: Rep[TS[T,:doc:boolean]], classify: (Rep[IndexVector]) =>
```

The same as confusionMatrix, except process testSamples as a batch.

```
 def crossValidate(dataSet: Rep[TS[T,:doc:boolean]], train: (Rep[TS[T,:doc:boolean]]) => Re
```

Compute a cross-validated score for the classifier using a user-specified metric from a cor

---

```
def crossValidateAUC(dataSet: Rep[TS[T,:doc:boolean]], train: (Rep[TS[T,:doc:boolean]]) =>
```

```
 def crossValidateAUCBatch(dataSet: Rep[TS[T,:doc:boolean]], train: (Rep[TS[T,:doc:boolean]
```

Same as crossValidateAUC, except with a batch of test samples at a time.

```
 def crossValidateBatch(dataSet: Rep[TS[T,:doc:boolean]], train: (Rep[TS[T,:doc:boolean]])
```

The same as crossValidate, except with a batch of test samples at a time.

```
 def crossValidateRaw(dataSet: Rep[TS[T,:doc:boolean]], train: (Rep[TS[T,:doc:boolean]]) =>
```

A generic cross validate routine that is shared by crossValidate and crossValidateBatch.

```
def fallout(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def fnr(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def fpr(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def fscore(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def holdOut(dataSet: Rep[TS[T,L]], pct: Rep[Double])(implicit ev0: TrainingSetLike[T,L,TS],
```

```
def holdOut2(dataSet: Rep[TS[T,L]], pctValidationSamples: Rep[Double], pctTestSamples: Rep
```

```
def precision(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def recall(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def sensitivity(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def specificity(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def tnr(x: Rep[DenseMatrix[Int]]): Rep[Double]
```

```
def tpr(x: Rep[DenseMatrix[Int]]): Rep[Double]
```