第三课 Spark ML数学基础

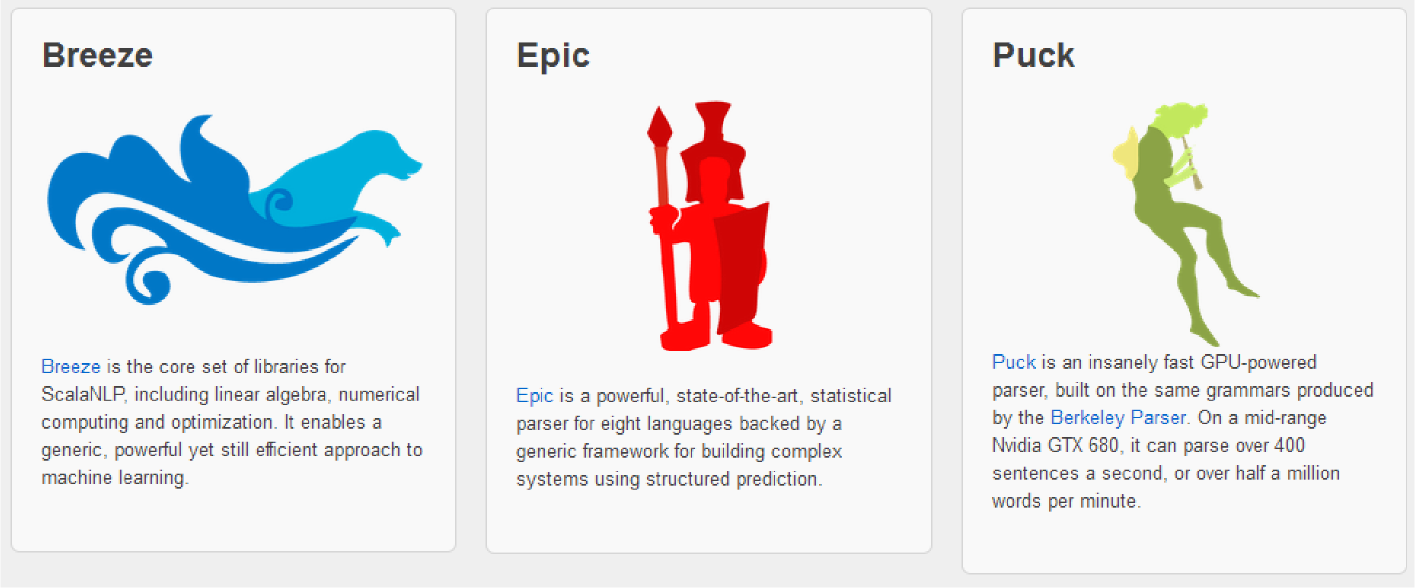
目录

目录

矩阵向量计算

Spark MLlib矩阵向量

- Spark MLlib底层的向量、矩阵运算使用了Breeze库，Breeze库提供了Vector/Matrix的实现以及相应计算的接口（Linalg）。但是在MLlib里面同时也提供了Vector和Linalg等的实现。



Breeze创建函数

- 在使用Breeze 库时，需要导入相关包：

```
import breeze.linalg._
import breeze.numerics._
```

- API：

http://www.scalanlp.org/api/breeze/index.html#breeze.linalg.package linalg - breeze.linalg

#ABCDEFGHIJKLMNOPQRSTUVWXYZ

display packages only

breeze.linalg hide focus

- *
- all
- any
- Axis
- BitVector
- Broadcasted
- BroadcastedColumns
- BroadcastedLike
- BroadcastedRows
- Broadcaster
- cholesky
- clip
- Counter
- Counter2
- Counter2Like
- CounterLike
- CSCMatrix
- DenseMatrix
- DenseVector
- det
- diag
- diagLowPrio
- diagLowPrio2
- diff
- eig
- eigSym
- fliplr
- flipud
- HashVector
- inv
- kron
- LinearAlgebraException
- logAndNormalize
- logDiff
- logNormalize

breeze

linalg

package **linalg**

This package contains everything relating to Vectors, Matrices, Tensors, etc.

If you're doing basic work, you probably want breeze.linalg.DenseVector and breeze.linalg.DenseMatrix, which support most operations. We also have breeze.linalg.SparseV
support for a sparse matrix (breeze.linalg.CSCMatrix).

This package object contains Matlab-esque functions for interacting with tensors and matrices.

▶ Linear Supertypes

Ordering Alphabetic By inheritance

Inherited linalg AnyRef Any

Hide All Show all Learn more about member selection

Visibility Public All

Type Members

sealed trait **Axis** extends AnyRef

This trait is commonly used for breeze.linalg.sum and its kin for summing along a particular axis of a Matrix.

class **BitVector** extends Vector[Boolean] with VectorLike[Boolean, BitVector]

TODO @ dwhl: What does enforceLength exactly do?

trait **Broadcasted**[+T, B] extends NumericOps[Broadcasted[T, B]]

TODO

case class **BroadcastedColumns**[T, B](underlying: T) extends BroadcastedLike[T, B, BroadcastedColumns[T, B]] with Product wit

Class for classes that are broadcasting their columns.

trait **BroadcastedLike**[T, B, Self <: Broadcasted[T, B]] extends Broadcasted[T, B] with NumericOps[Self]

case class **BroadcastedRows**[T, RowType](underlying: T) extends BroadcastedLike[T, RowType, BroadcastedRows[T, RowType]] with
Serializable

Class for classes that are broadcasting their rows.

Breeze创建函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
全0矩阵	DenseMatrix.zeros[Double](n,m)	zeros(n,m)	zeros((n,m))
全0向量	DenseVector.zeros[Double](n)	zeros(n)	zeros(n)
全1向量	DenseVector.ones[Double](n)	ones(n)	ones(n)
按数值填充向量	DenseVector.fill(n){5.0}	ones(n) * 5	ones(n) * 5
生成随机向量	DenseVector.range(start,stop,step) orVector.rangeD(start,stop,step)		
线性等分向量（用于产生start,stop之间的N点行矢量）	DenseVector.linspace(start,stop,numvals)	linspace(0,20,15)	
单位矩阵	DenseMatrix.eye[Double](n)	eye(n)	eye(n)
对角矩阵	diag(DenseVector(1.0,2.0,3.0))	diag([1 2 3])	diag((1,2,3))
按照行创建矩阵	DenseMatrix((1.0,2.0), (3.0,4.0))	[1 2; 3 4]	array([[1,2], [3,4]])
按照行创建向量	DenseVector(1,2,3,4)	[1 2 3 4]	array([1,2,3,4])
向量转置	DenseVector(1,2,3,4).t	[1 2 3 4]'	array([1,2,3]).reshape(-1,1)
从函数创建向量	DenseVector.tabulate(3){i => 2*i}		
从函数创建矩阵	DenseMatrix.tabulate(3, 2){case (i, j) => i+j}		
从数组创建向量	new DenseVector(Array(1, 2, 3, 4))		
从数组创建矩阵	new DenseMatrix(2, 3, Array(11, 12, 13, 21, 22, 23))		
0 到 1的随机向量	DenseVector.rand(4)		
0 到 1的随机矩阵	DenseMatrix.rand(2, 3)		

Breeze创建函数

```
scala> val m1 = DenseMatrix.zeros[Double](2,3)
m1: breeze.linalg.DenseMatrix[Double] =
0.0  0.0  0.0
0.0  0.0  0.0

scala> val v1 = DenseVector.zeros[Double](3)
v1: breeze.linalg.DenseVector[Double] = DenseVector(0.0, 0.0, 0.0)

scala> val v2 = DenseVector.ones[Double](3)
v2: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 1.0, 1.0)

scala> val v3 = DenseVector.fill(3){5.0}
v3: breeze.linalg.DenseVector[Double] = DenseVector(5.0, 5.0, 5.0)
```

Breeze创建函数

```
scala> val v4 = DenseVector.range(1,10,2)
v4: breeze.linalg.DenseVector[Int] = DenseVector(1, 3, 5, 7, 9)

scala> val m2 = DenseMatrix.eye[Double](3)
m2: breeze.linalg.DenseMatrix[Double] =
1.0  0.0  0.0
0.0  1.0  0.0
0.0  0.0  1.0

scala> val v6 = diag(DenseVector(1.0,2.0,3.0))
v6: breeze.linalg.DenseMatrix[Double] =
1.0  0.0  0.0
0.0  2.0  0.0
0.0  0.0  3.0
```

Breeze创建函数

```
scala> val v8 = DenseVector(1,2,3,4)
v8: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4)

scala> val v9 = DenseVector(1,2,3,4).t
v9: breeze.linalg.Transpose[breeze.linalg.DenseVector[Int]] = Transpose(DenseVector(1, 2, 3, 4))

scala> val v10 = DenseVector.tabulate(3){i => 2*i}
v10: breeze.linalg.DenseVector[Int] = DenseVector(0, 2, 4)

scala> val m4 = DenseMatrix.tabulate(3, 2){case (i, j) => i+j}
m4: breeze.linalg.DenseMatrix[Int] =
0  1
1  2
2  3
```

Breeze创建函数

```
scala> val v11 = new DenseVector(Array(1, 2, 3, 4))
v11: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4)
scala> val m5 = new DenseMatrix(2, 3, Array(11, 12, 13, 21, 22, 23))
m5: breeze.linalg.DenseMatrix[Int] =
11  13  22
12  21  23
scala> val v12 = DenseVector.rand(4)
v12: breeze.linalg.DenseVector[Double] = DenseVector(0.7517657487447951, 0.8171495400874123, 0.8923542318540489,
0.17431125994911 9)
scala> val m6 = DenseMatrix.rand(2, 3)
m6: breeze.linalg.DenseMatrix[Double] =
0.5349430131148125   0.8822136832272578  0.7946323804433382
0.41097756311601086  0.3181490074596882  0.34195102205697414
```

Breeze元素访问

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
指定位置	a(0,1)	a(1,2)	a[0,1]
向量子集	a(1 to 4) or a(1 until 5) ora.slice(1,5)	a(2:5)	a[1:5]
按照指定步长取子集	a(5 to 0 by -1)	a(6:-1:1)	a[5:0:-1]
指定开始位置至结尾	a(1 to -1)	a(2:end)	a[1:]
最后一个元素	a(-1)	a(end)	a[-1]
矩阵指定列	a(::, 2)	a(:,3)	a[:,2]

Breeze元素访问

```
scala> val a = DenseVector(1,2,3,4,5,6,7,8,9,10)
a: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> a(0)
res2: Int = 1

scala> a(1 to 4)
res4: breeze.linalg.DenseVector[Int] = DenseVector(2, 3, 4, 5)

scala> a(5 to 0 by -1)
res5: breeze.linalg.DenseVector[Int] = DenseVector(6, 5, 4, 3, 2, 1)

scala> a(1 to -1)
res6: breeze.linalg.DenseVector[Int] = DenseVector(2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Breeze元素访问

```
scala> a( -1 )
res7: Int = 10

scala> val m = DenseMatrix((1.0,2.0,3.0), (3.0,4.0,5.0))
m: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
3.0  4.0  5.0

scala> m(0,1)
res8: Double = 2.0

scala> m(::,1)
res9: breeze.linalg.DenseVector[Double] = DenseVector(2.0, 4.0)
```

Breeze元素操作

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
调整矩阵形状	a.reshape(3, 2)	reshape(a, 3, 2)	a.reshape(3,2)
矩阵转成向量	a.toDenseVector (Makes copy)	a(:)	a.flatten()
复制下三角	lowerTriangular(a)	tril(a)	tril(a)
复制上三角	upperTriangular(a)	triu(a)	triu(a)
矩阵复制	a.copy		np.copy(a)
取对象线元素	diag(a)	NA	diagonal(a)(Numpy >= 1.9)
子集赋数值	a(1 to 4) := 5.0	a(2:5) = 5	a[1:4] = 5
子集赋向量	a(1 to 4) := DenseVector(1.0,2.0,3.0)	a(2:5) = [1 2 3]	a[1:4] = array([1,2,3])
矩阵赋值	a(1 to 3,1 to 3) := 5.0	a(2:4,2:4) = 5	a[1:3,1:3] = 5
矩阵列赋值	a(::, 2) := 5.0	a(:,3) = 5	a[:,2] = 5
垂直连接矩阵	DenseMatrix.vertcat(a,b)	[a ; b]	vstack((a,b))
横向连接矩阵	DenseMatrix.horzcat(d,e)	[a , b]	hstack((a,b))
向量连接	DenseVector.vertcat(a,b)	[a b]	concatenate((a,b))

Breeze元素操作

```
scala> val m = DenseMatrix((1.0,2.0,3.0), (3.0,4.0,5.0))
m: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
3.0  4.0  5.0

scala> m.reshape(3, 2)
res11: breeze.linalg.DenseMatrix[Double] =
1.0  4.0
3.0  3.0
2.0  5.0

scala>m.toDenseVector
res12: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 3.0, 2.0, 4.0, 3.0, 5.0)
```

Breeze元素操作

```
scala> val m = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0) , (7.0,8.0,9.0))
m: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0
```

```
scala> val m = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0) , (7.0,8.0,9.0))
m: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0
```

Breeze元素操作

```
scala> lowerTriangular(m)
res19: breeze.linalg.DenseMatrix[Double] =
1.0  0.0  0.0
4.0  5.0  0.0
7.0  8.0  9.0

scala> upperTriangular(m)
res20: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
0.0  5.0  6.0
0.0  0.0  9.0
```

Breeze元素操作

```
scala>  m.copy
res21: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0

scala> diag(m)
res22: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 5.0, 9.0)

scala> m(::, 2) := 5.0
res23: breeze.linalg.DenseVector[Double] = DenseVector(5.0, 5.0, 5.0)
```

Breeze元素操作

```
scala> m
res24: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  5.0
4.0  5.0  5.0
7.0  8.0  5.0
scala> m(1 to 2,1 to 2) := 5.0
res32: breeze.linalg.DenseMatrix[Double] =
5.0  5.0
5.0  5.0
scala> m
res33: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  5.0
4.0  5.0  5.0
7.0  5.0  5.0
```

Breeze元素操作

```
scala> val a = DenseVector(1,2,3,4,5,6,7,8,9,10)
a: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
scala> a(1 to 4) := 5
res27: breeze.linalg.DenseVector[Int] = DenseVector(5, 5, 5, 5)
scala> a(1 to 4) := DenseVector(1,2,3,4)
res29: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4)

scala> a
res30: breeze.linalg.DenseVector[Int] = DenseVector(1, 1, 2, 3, 4, 6, 7, 8, 9, 10)

scala> val a1 = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0))
a1: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
```

Breeze元素操作

```
scala> val a2 = DenseMatrix((1.0,1.0,1.0), (2.0,2.0,2.0))
a2: breeze.linalg.DenseMatrix[Double] =
1.0  1.0  1.0
2.0  2.0  2.0
scala> DenseMatrix.vertcat(a1,a2)
res34: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
1.0  1.0  1.0
2.0  2.0  2.0
scala> DenseMatrix.horzcat(a1,a2)
res35: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0  1.0  1.0  1.0
4.0  5.0  6.0  2.0  2.0  2.0
```

Breeze元素操作

```
scala> val b1 = DenseVector(1,2,3,4)
b1: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4)

scala> val b2 = DenseVector(1,1,1,1)
b2: breeze.linalg.DenseVector[Int] = DenseVector(1, 1, 1, 1)

scala> DenseVector.vertcat(b1,b2)
res36: breeze.linalg.DenseVector[Int] = DenseVector(1, 2, 3, 4, 1, 1, 1, 1)
```

Breeze数值计算函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
元素加法	a + b	a + b	a + b
元素乘法	a :* b	a .* b	a * b
元素除法	a :/ b	a ./ b	a / b
元素比较	a :< b	a < b	a < b
元素相等	a :== b	a == b	a == b
元素追加	a :+= 1.0	a += 1	a += 1
元素追乘	a :*= 2.0	a *= 2	a *= 2
向量点积	a dot b,a.t * b†	dot(a,b)	dot(a,b)
元素最大值	max(a)	max(a)	a.max()
元素最大值及位置	argmax(a)	[v i] = max(a); i	a.argmax()

Breeze数值计算函数

```
scala> val a = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0))
a: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0

scala>  val b = DenseMatrix((1.0,1.0,1.0), (2.0,2.0,2.0))
b: breeze.linalg.DenseMatrix[Double] =
1.0  1.0  1.0
2.0  2.0  2.0

scala> a + b
res37: breeze.linalg.DenseMatrix[Double] =
2.0  3.0  4.0
6.0  7.0  8.0
```

Breeze数值计算函数

```
scala> a :* b
res38: breeze.linalg.DenseMatrix[Double] =
1.0  2.0   3.0
8.0  10.0  12.0

scala> a :/ b
res39: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
2.0  2.5  3.0

scala> a :< b
res40: breeze.linalg.DenseMatrix[Boolean] =
false  false  false
false  false  false
```

Breeze数值计算函数

```
scala> a :== b
res41: breeze.linalg.DenseMatrix[Boolean] =
true   false  false
false  false  false

scala> a :+= 1.0
res42: breeze.linalg.DenseMatrix[Double] =
2.0  3.0  4.0
5.0  6.0  7.0

scala> a :*= 2.0
res43: breeze.linalg.DenseMatrix[Double] =
4.0   6.0   8.0
10.0  12.0  14.0
```

Breeze数值计算函数

```
scala> max(a)
res47: Double = 14.0

scala> argmax(a)
res48: (Int, Int) = (1,2)

scala> DenseVector(1, 2, 3, 4) dot DenseVector(1, 1, 1, 1)
res50: Int = 10
```

Breeze求和函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
元素求和	sum(a)	sum(sum(a))	a.sum()
每一列求和	sum(a, Axis._0) orsum(a(::, *))	sum(a)	sum(a,0)
每一行求和	sum(a, Axis._1) orsum(a(*, ::))	sum(a')	sum(a,1)
对角线元素和	trace(a)	trace(a)	a.trace()
累积和	accumulate(a)	cumsum(a)	a.cumsum()

Breeze求和函数

```
scala> val a = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0) , (7.0,8.0,9.0))
a: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0

scala> sum(a)
res51: Double = 45.0

scala> sum(a, Axis._0)
res52: breeze.linalg.DenseMatrix[Double] = 12.0  15.0  18.0
```

Breeze求和函数

```
scala> sum(a, Axis._1)
res53: breeze.linalg.DenseVector[Double] = DenseVector(6.0, 15.0, 24.0)

scala> trace(a)
res54: Double = 15.0

scala> accumulate(DenseVector(1, 2, 3, 4))
res56: breeze.linalg.DenseVector[Int] = DenseVector(1, 3, 6, 10)
```

Breeze布尔函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
元素与操作	a :& b	a && b	a & b
元素或操作	a :| b	a || b	a | b
元素非操作	!a	~a	~a
任意元素非零	any(a)	any(a)	any(a)
所有元素非零	all(a)	all(a)	all(a)

Breeze布尔函数

```
scala> val a = DenseVector(true, false, true)
a: breeze.linalg.DenseVector[Boolean] = DenseVector(true, false, true)

scala> val b = DenseVector(false, true, true)
b: breeze.linalg.DenseVector[Boolean] = DenseVector(false, true, true)

scala> a :& b
res57: breeze.linalg.DenseVector[Boolean] = DenseVector(false, false, true)

scala> a :| b
res58: breeze.linalg.DenseVector[Boolean] = DenseVector(true, true, true)

scala> !a
res59: breeze.linalg.DenseVector[Boolean] = DenseVector(false, true, false)
```

Breeze布尔函数

```
scala> val a = DenseVector(1.0, 0.0, -2.0)
a: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 0.0, -2.0)

scala> any(a)
res60: Boolean = true

scala> all(a)
res61: Boolean = false
```

Breeze线性代数函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
线性求解	a \ b	a \ b	linalg.solve(a,b)
转置	a.t	a'	a.conj.transpose()
求特征值	det(a)	det(a)	linalg.det(a)
求逆	inv(a)	inv(a)	linalg.inv(a)
求伪逆	pinv(a)	pinv(a)	linalg.pinv(a)
求范数	norm(a)	norm(a)	norm(a)
特征值和特征向量	eigSym(a)	[v,l] = eig(a)	linalg.eig(a)[0]
特征值	val (er, ei, _) = eig(a) (实部与虚部分开)	eig(a)	linalg.eig(a)[0]
特征向量	eig(a)._3	[v,l] = eig(a)	linalg.eig(a)[1]
奇异值分解	val svd.SVD(u,s,v) = svd(a)	svd(a)	linalg.svd(a)
求矩阵的秩	rank(a)	rank(a)	rank(a)
矩阵长度	a.length	size(a)	a.size
矩阵行数	a.rows	size(a,1)	a.shape[0]
矩阵列数	a.cols	size(a,2)	a.shape[1]

Breeze线性代数函数

```
scala> val a = DenseMatrix((1.0,2.0,3.0), (4.0,5.0,6.0) , (7.0,8.0,9.0))
a: breeze.linalg.DenseMatrix[Double] =
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0

scala> val b = DenseMatrix((1.0,1.0,1.0), (1.0,1.0,1.0) , (1.0,1.0,1.0))
b: breeze.linalg.DenseMatrix[Double] =
1.0  1.0  1.0
1.0  1.0  1.0
1.0  1.0  1.0
```

Breeze线性代数函数

```
scala> a \ b
res74: breeze.linalg.DenseMatrix[Double] =
-2.5  -2.5  -2.5
4.0   4.0   4.0
-1.5  -1.5  -1.5

scala> a.t
res63: breeze.linalg.DenseMatrix[Double] =
1.0  4.0  7.0
2.0  5.0  8.0
3.0  6.0  9.0

scala> det(a)
res64: Double = 6.661338147750939E-16
```

Breeze线性代数函数

```
scala> a \ b
res74: breeze.linalg.DenseMatrix[Double] =
-2.5  -2.5  -2.5
4.0   4.0   4.0
-1.5  -1.5  -1.5

scala> a.t
res63: breeze.linalg.DenseMatrix[Double] =
1.0  4.0  7.0
2.0  5.0  8.0
3.0  6.0  9.0

scala> det(a)
res64: Double = 6.661338147750939E-16
```

Breeze取整函数

操作名称	Breeze函数	对应Matlab函数	对应Numpy函数
四舍五入	round(a)	round(a)	around(a)
最小整数	ceil(a)	ceil(a)	ceil(a)
最大整数	floor(a)	floor(a)	floor(a)
符号函数	signum(a)	sign(a)	sign(a)
取正数	abs(a)	abs(a)	abs(a)

Breeze取整函数

```
scala> val a = DenseVector(1.2, 0.6, -2.3)
a: breeze.linalg.DenseVector[Double] = DenseVector(1.2, 0.6, -2.3)
scala> round(a)
res75: breeze.linalg.DenseVector[Long] = DenseVector(1, 1, -2)
scala> ceil(a)
res76: breeze.linalg.DenseVector[Double] = DenseVector(2.0, 1.0, -2.0)
scala> floor(a)
res77: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 0.0, -3.0)

scala> signum(a)
res78: breeze.linalg.DenseVector[Double] = DenseVector(1.0, 1.0, -1.0)

scala> abs(a)
res79: breeze.linalg.DenseVector[Double] = DenseVector(1.2, 0.6, 2.3)
```

Breeze其它函数

- Breeze三角函数

Breeze三角函数包括：

sin, sinh, asin, asinh

cos, cosh, acos, acosh

tan, tanh, atan, atanh

atan2

sinc(x) ，即sin(x)/x

sincpi(x) ，即 sinc(x * Pi)

- Breeze对数和指数函数

Breeze对数和指数函数包括：

log, exp log10

log1p, expm1

sqrt, sbrt

pow

BLAS介绍

- BLAS按照功能被分为三个级别：
- Level 1：矢量-矢量运算，比如点积（ddot)，加法和数乘 (daxpy)， 绝对值的和（dasum)，等等；
- Level 2：矩阵-矢量运算，最重要的函数是一般的矩阵向量乘法(dgemv)；
- Level 3：矩阵-矩阵运算，最重要的函数是一般的矩阵乘法 (dgemm)；
- 每一种函数操作都区分不同数据类型（单精度、双精度、复数）

Level 1 BLAS

	dim	scalar	vector	vector	scalars	5-element array		prefixes
SUBROUTINE xROTG (					A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(					D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,	C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		X, INCX,	Y, INCY,		PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (	N,		X, INCX,	Y, INCY)			$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (	N,	ALPHA,	X, INCX)				$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (	N,		X, INCX,	Y, INCY)			$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)			$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (	N,		X, INCX)				$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
FUNCTION xASUM (	N,		X, INCX)				$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(	N,		X, INCX)				$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

	options	dim	b-width	scalar	matrix	vector	scalar	vector		prefixes
xGEMV (	TRANS,	M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (	TRANS,	M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (	UPLO,	N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (	UPLO,	N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (	UPLO, TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV (	UPLO, TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV (	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z

	options	dim	scalar	vector	vector	matrix		prefixes
xGER (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^T + A$	S, D
xSPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^T + A$	S, D

BLAS介绍

3.2.1 BLAS 向量-向量运算

单精度类型的向量-向量运算函数如下：

- SROTG——Givens 旋转设置
- SROTMG——改进 Givens 旋转设置
- SROT——Givens 旋转
- SROTM——改进 Givens 旋转
- SSWAP——交换 x 和 y
- SSCAL——常数 a 乘以向量 x()
- SCOPY——把 x 复制到 y
- SAXPY——向量 y+常数 a 乘以向量 x (y = a*x + y)
- SDOT——点积
- SDSDOT——扩展精度累积的点积
- SNRM2——欧氏范数
- SCNRM2——欧氏范数
- SASUM——绝对值之和
- ISAMAX——最大值位置

BLAS介绍

3.2.2 BLAS 矩阵-向量运算

- SGEMV——矩阵向量乘法
- SGBMV——带状矩阵向量乘法
- SSYMV——对称矩阵向量乘法
- SSBMV——对称带状矩阵向量乘法
- SSPMV——对称填充矩阵向量乘法
- STRMV——三角矩阵向量乘法
- STBMV——三角带状矩阵向量乘法
- STPMV——三角填充矩阵向量乘法
- STRSV——求解三角矩阵
- STBSV——求解三角带状矩阵
- STPSV——求解三角填充矩阵
- SGER——A := alpha*x*y' + A
- SSYR——A := alpha*x*x' + A
- SSPR——A := alpha*x*x' + A
- SSYR2——A := alpha*x*y' + alpha*y*x' + A
- SSPR2——A := alpha*x*y' + alpha*y*x' + A

3.2.3 BLAS 矩阵-矩阵运算

单精度类型的矩阵-矩阵运算函数如下：

- SGEMM——矩阵乘法
- SSYMM——对称矩阵乘法
- SSYRK——对称矩阵的秩-k 修正
- SSYR2K——对称矩阵的秩-2k 修正
- STRMM——三角矩阵乘法
- STRSM——多重右端的三角线性方程组求解

目录

分类效果评估指标

分类效果指标

		True class p	True class n
Hypothesized class	Y	True Positives	False Positives
	N	False Negatives	True Negatives
Column totals:		P	N

$$\text{fp rate} = \frac{FP}{N} \qquad \text{tp rate} = \frac{TP}{P}$$

$$\text{precision} = \frac{TP}{TP+FP} \qquad \text{recall} = \frac{TP}{P}$$

$$\text{accuracy} = \frac{TP+TN}{P+N}$$

$$\text{F-measure} = \frac{2}{1/\text{precision}+1/\text{recall}}$$

Fig. 1. Confusion matrix and common performance metrics calculated from it.

精确率，precision = TP / (TP + FP)
模型判为正的所有样本中有多少是真正的正样本

召回率，recall = TP / (P)

准确率，accuracy = (TP + TN) / (P+ N)

真正类（True positive）：实例是正类并且被预测成正类
假正类（False positive）：实例是负类并且被预测成正类
真负类（True negative）：实例是负类并且被预测成负类
假负类（false negative）：实例是正类并且被预测成负类

综合评价指标（F-Measure）

- P和R指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）。

$$F = \frac{(\alpha^2 + 1)P * R}{\alpha^2 (P + R)}$$

- **F-Measure是Precision和Recall加权调和平均**：
- 当参数α=1时，就是最常见的F1，也即

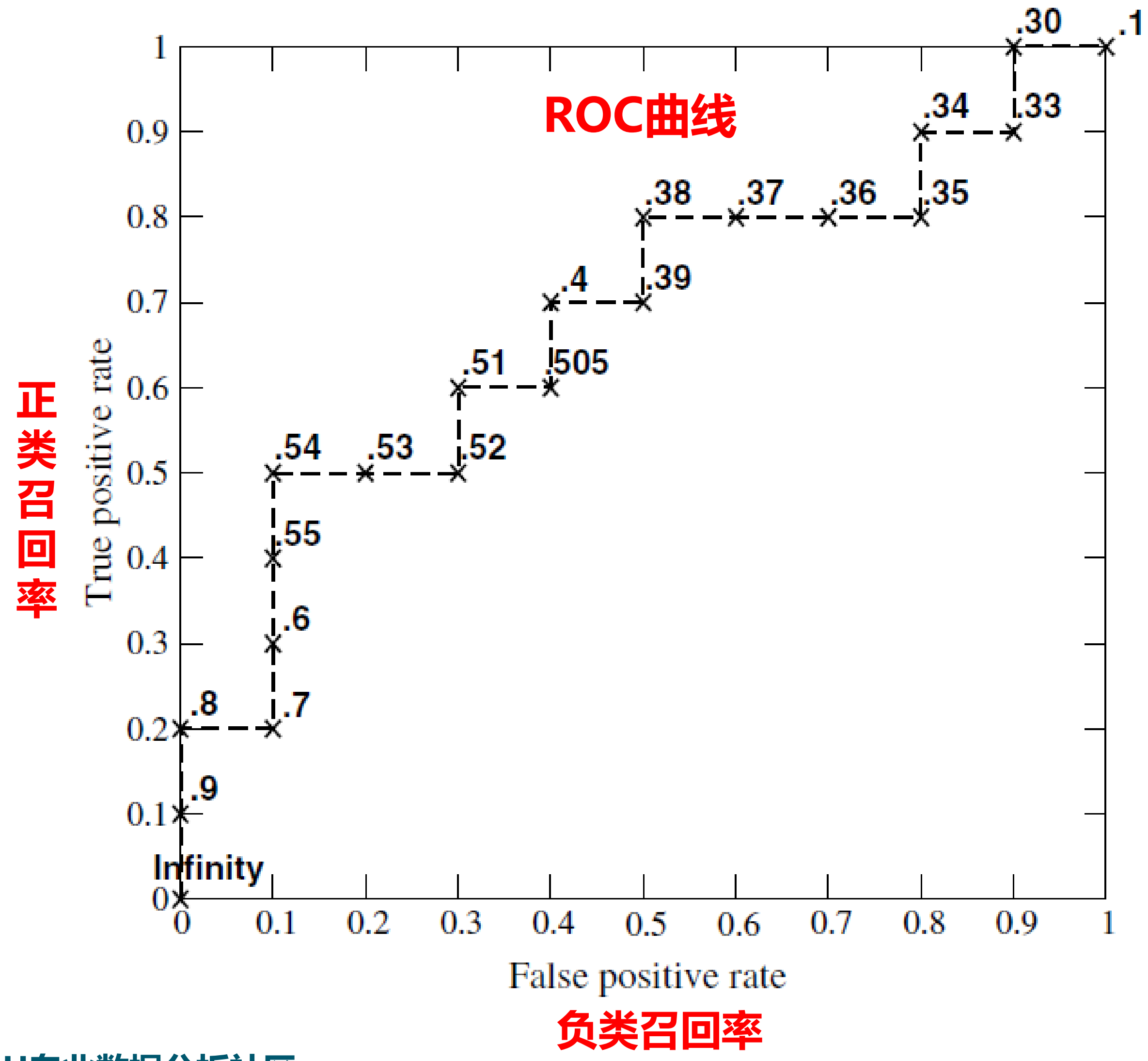
$$F1 = \frac{2 * P * R}{P + R}$$

- 可知F1综合了P和R的结果，当F1较高时则能说明试验方法比较有效。

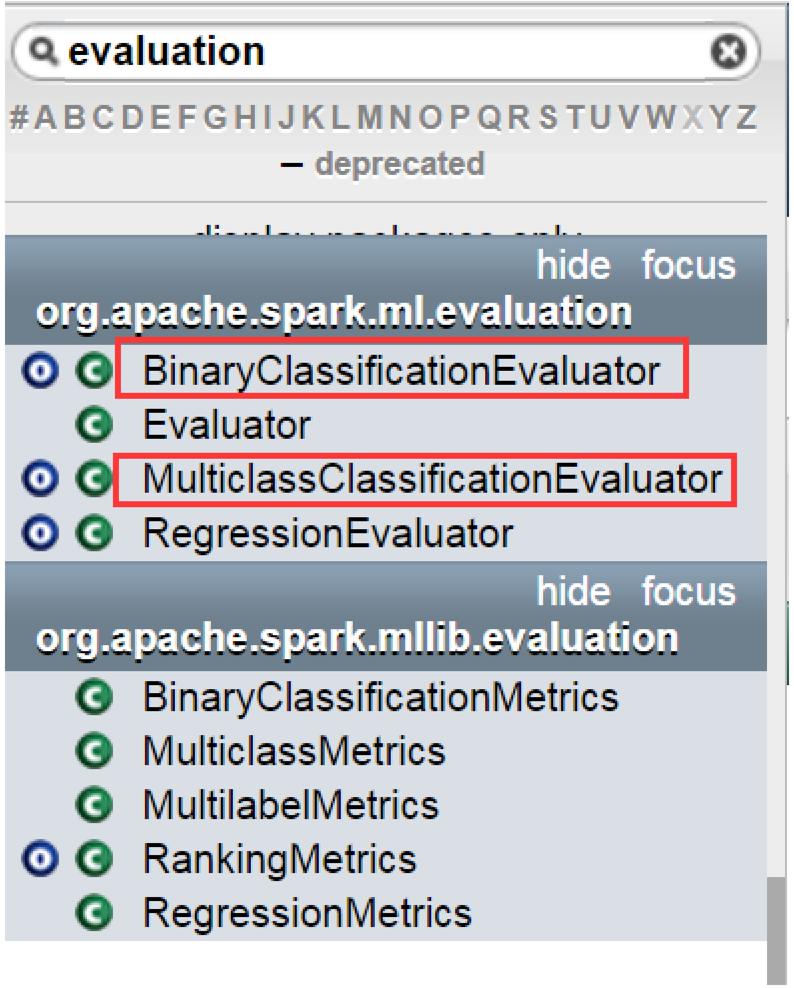
分类效果指标

10个正类，10个负类；按照score降序排列

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



分类效果指标


evaluation
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
– deprecated
hide focus
org.apache.spark.ml.evaluation
BinaryClassificationEvaluator
Evaluator
MulticlassClassificationEvaluator
RegressionEvaluator
hide focus
org.apache.spark.mllib.evaluation
BinaryClassificationMetrics
MulticlassMetrics
MultilabelMetrics
RankingMetrics
RegressionMetrics

分类效

org.apache.spark.ml.evaluation

BinaryClassificationEvaluator

Related Docs

`class BinaryClassificationEvaluator extends Evaluator with HasRawPredictionCol with HasLabelCol with DefaultParamsW`

Evaluator for binary classification, which expects two input columns: rawPrediction and label. The rawPrediction column can be of type double (binary 0 1) or of type vector (length-2 vector of raw predictions, scores, or label probabilities).

Annotations	@Since("1.2.0") @Experimental()
Source	BinaryClassificationEvaluator.scala

▶ Linear Supertypes

Ordering Grouped Alphabetic By Inheritance

Inherited BinaryClassificationEvaluator DefaultParamsWritable MLWritable HasLabelCol HasRawPredictionCol Evaluator Params AnyRef Any

Hide All Show All

Visibility Public All

Parameters

A list of (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

`final val labelCol: Param[String]`

Param for label column name.

`val metricName: Param[String]`

param for metric name in evaluation $\boxed{\text{(supports "areaUnderROC" (default), "areaUnderPR")}}$

Annotations @Since("1.2.0")

`final val rawPredictionCol: Param[String]`

Param for raw prediction (a.k.a.

org.apache.spark.ml.evaluation

MulticlassClassificationEvaluator

Related Docs: object Mu

`class MulticlassClassificationEvaluator extends Evaluator with HasPredictionCol with HasLabelCol with DefaultParamsWritabl`

Evaluator for multiclass classification, which expects two input columns: prediction and label.

Annotations	@Since("1.5.0") @Experimental()
Source	MulticlassClassificationEvaluator.scala

▸ Linear Supertypes

Ordering Grouped Alphabetic By Inheritance

Inherited MulticlassClassificationEvaluator DefaultParamsWritable MLWritable HasLabelCol HasPredictionCol Evaluator Params Seriali

AnyRef Any

Hide All Show All

Visibility Public All

Parameters

A list of (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

`final val labelCol: Param[String]`

Param for label column name.

`val metricName: Param[String]`

param for metric name in evaluation (supports "f1" (default), "weightedPrecision", "weightedRecall", "accuracy")

`final val predictionCol: Param[String]`

Param for prediction column name.

分类效果指标

ATAGURU 炼数成金

```scala
// 正确率
val evaluator1 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("accuracy")
val accuracy = evaluator1.evaluate(predictions)
println("Test Error = " + (1.0 - accuracy))
// f1
val evaluator2 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("f1")
val f1 = evaluator2.evaluate(predictions)
println("f1 = " + f1)
```

分类效果指标

ATAGURU 炼数成金

```
// Precision
val evaluator3 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("weightedPrecision")
val Precision = evaluator3.evaluate(predictions)
println("Precision = " + Precision)
// Recall
val evaluator4 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("weightedRecall")
val Recall = evaluator4.evaluate(predictions)
println("Recall = " + Recall)
```

分类效果指标

```
// AUC
val evaluator5 = new BinaryClassificationEvaluator().
  setLabelCol("indexedLabel").
  setRawPredictionCol("prediction").
  setMetricName("areaUnderROC")
val AUC = evaluator5.evaluate(predictions)
println("Test AUC = " + AUC)
// aupr
val evaluator6 = new BinaryClassificationEvaluator().
  setLabelCol("indexedLabel").
  setRawPredictionCol("prediction").
  setMetricName("areaUnderPR")
val aupr = evaluator6.evaluate(predictions)
println("Test aupr = " + aupr)
```

目录

交叉-验证方法

交叉验证法

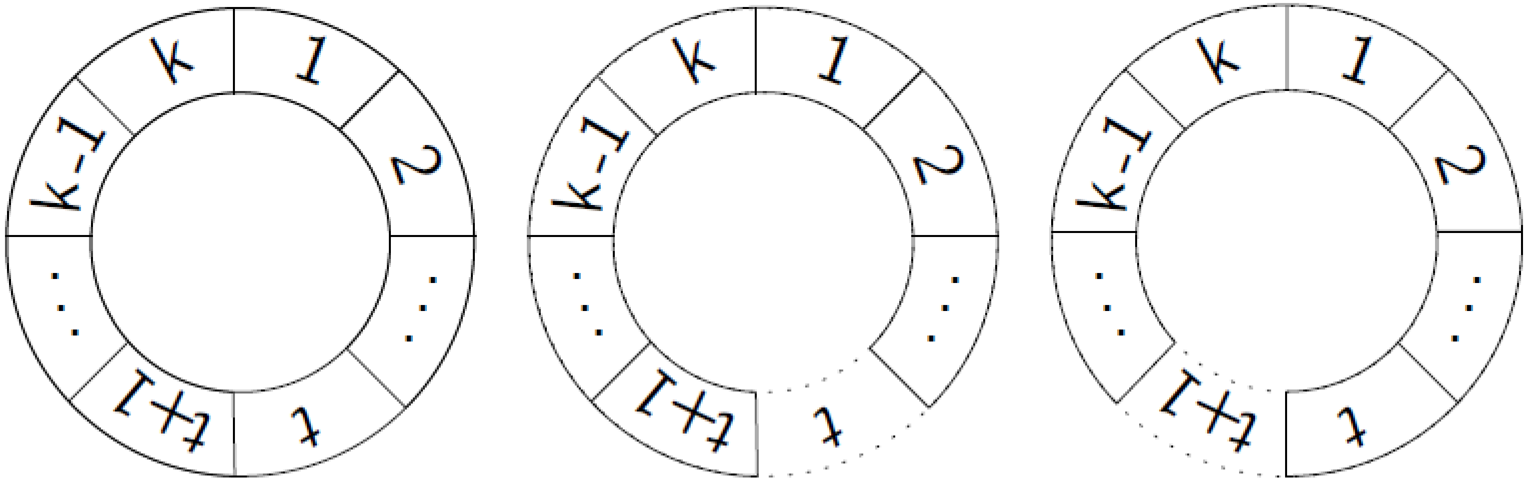
1、“交叉验证法”（cross validation）先将数据集D划分为k个大小相似的互斥子集，即D=D1并D2并D3...并Dk，每个子集之间没有交集。

2、然后每次用k-1个子集的并集作为训练集，余下的那个作为测试集，这样得到k组训练/测试集。

3、可以进行k次训练和测试，最终返回的是这k个结果的均值。

4、可以随机使用不同的划分多次，比如10次10折交叉验证通常把交叉验证法称为“k折交叉验证”（k-fold cross validation），k最常用的取值是10，为10折交叉验证

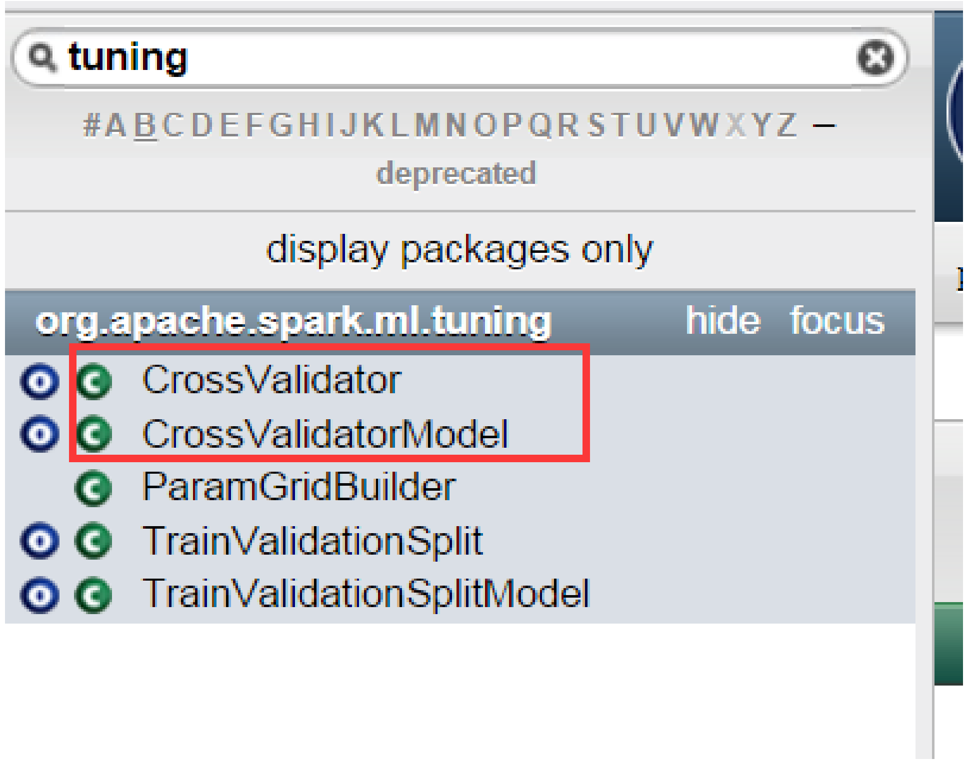
交叉-验证方法



(a) k subsets (b) t^{th} as test (c) $(t+1)^{\text{th}}$ as test

Figure: k-fold cross-validation

交叉-验证方法


tuning
#ABCDEFGHIJKLMNOPQRSTUVWXYZ –
deprecated
display packages only
org.apache.spark.ml.tuning
hide focus
CrossValidator
CrossValidatorModel
ParamGridBuilder
TrainValidationSplit
TrainValidationSplitModel

交叉-验证方法

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.{ HashingTF, Tokenizer }
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.tuning.{ CrossValidator, ParamGridBuilder }
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

交叉-验证方法

```scala
// 样本数据，格式为(id, text, label).
  val training = spark.createDataFrame(Seq(
    (0L, "a b c d e spark", 1.0),
    (1L, "b d", 0.0),
    (2L, "spark f g h", 1.0),
    (3L, "hadoop mapreduce", 0.0),
    (4L, "b spark who", 1.0),
    (5L, "g d a y", 0.0),
    (6L, "spark fly", 1.0),
    (7L, "was mapreduce", 0.0),
    (8L, "e spark program", 1.0),
    (9L, "a e c l", 0.0),
    (10L, "spark compile", 1.0),
    (11L, "hadoop software", 0.0))).toDF("id", "text", "label")
```

交叉-验证方法

```scala
// 建立ML管道，包括: tokenizer, hashingTF, and lr.
val tokenizer = new Tokenizer()
  .setInputCol("text")
  .setOutputCol("words")
val hashingTF = new HashingTF()
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("features")
val lr = new LogisticRegression()
  .setMaxIter(10)
val pipeline = new Pipeline()
  .setStages(Array(tokenizer, hashingTF, lr))

// 采用ParamGridBuilde方法来建立网格搜索.
// 网格的参数包括：hashingTF.numFeatures 3个参数， lr.regParam 2个参数
// 网格总共大小为： 3 x 2 = 6，采用交叉验证来选择最优参数.
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 100, 1000))
  .addGrid(lr.regParam, Array(0.1, 0.01))
  .build()
```

交叉-验证方法

```
// 建立一个交叉验证的评估器，设置评估器的参数
  val cv = new CrossValidator()
    .setEstimator(pipeline)
    .setEvaluator(new BinaryClassificationEvaluator)
    .setEstimatorParamMaps(paramGrid)
    .setNumFolds(2) // Use 3+ in practice

  // 运行交叉验证评估器，得到最佳参数集的模型.
  val cvModel = cv.fit(training)
```

交叉-验证方法

```scala
// 测试数据.
val test = spark.createDataFrame(Seq(
  (4L, "spark i j k"),
  (5L, "l m n"),
  (6L, "mapreduce spark"),
  (7L, "apache hadoop"))).toDF("id", "text")

// 测试， cvModel会选择最佳lrModel进行预测.
cvModel.transform(test)
  .select("id", "text", "probability", "prediction")
  .collect()
  .foreach {
    case Row(id: Long, text: String, prob: Vector, prediction: Double) =>
      println(s"($id, $text) --> prob=$prob, prediction=$prediction")
  }
```

交叉-验证方法

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.{ HashingTF, Tokenizer }
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.tuning.{ CrossValidator, ParamGridBuilder }
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.feature.{HashingTF, Tokenizer}
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

交叉-

```
// 样本数据，格式为(id, text, label).
val training = spark.createDataFrame(Seq(
  (0L, "a b c d e spark", 1.0),
  (1L, "b d", 0.0),
  (2L, "spark f g h", 1.0),
  (3L, "hadoop mapreduce", 0.0),
  (4L, "b spark who", 1.0),
  (5L, "g d a y", 0.0),
  (6L, "spark fly", 1.0),
  (7L, "was mapreduce", 0.0),
  (8L, "e spark program", 1.0),
  (9L, "a e c l", 0.0),
  (10L, "spark compile", 1.0),
  (11L, "hadoop software", 0.0))).toDF("id", "text", "label")

  training.show()
```

```
training: org.apache.spark.sql.DataFrame = [id: bigint, text: string ... 1 more field]
+---+----------------+-----+
| id|            text|label|
+---+----------------+-----+
|  0| a b c d e spark|  1.0|
|  1|             b d|  0.0|
|  2|     spark f g h|  1.0|
|  3|hadoop mapreduce|  0.0|
|  4|     b spark who|  1.0|
|  5|         g d a y|  0.0|
|  6|       spark fly|  1.0|
|  7|   was mapreduce|  0.0|
|  8| e spark program|  1.0|
```

交叉-验证方法

```
// 建立ML管道, 包括: tokenizer, hashingTF, and lr.
val tokenizer = new Tokenizer()
  .setInputCol("text")
  .setOutputCol("words")
val hashingTF = new HashingTF()
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("features")
val lr = new LogisticRegression()
  .setMaxIter(10)
val pipeline = new Pipeline()
  .setStages(Array(tokenizer, hashingTF, lr))
```

```
tokenizer: org.apache.spark.ml.feature.Tokenizer = tok_25c9f8d9cc39
hashingTF: org.apache.spark.ml.feature.HashingTF = hashingTF_5a96e1199390
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_d12da15c7c9d
pipeline: org.apache.spark.ml.Pipeline = pipeline_68b09304033f
```

交叉

```
// 采用ParamGridBuilde方法来建立网格搜索.
// 网格的参数包括: hashingTF.numFeatures 3个参数,  lr.regParam 2个参数
// 网格总共大小为:  3 x 2 = 6, 采用交叉验证来选择最优参数.
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 100, 1000))
  .addGrid(lr.regParam, Array(0.1, 0.01))
  .build()
```

```
paramGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        hashingTF_5a96e1199390-numFeatures: 10,
        logreg_d12da15c7c9d-regParam: 0.1
}, {
        hashingTF_5a96e1199390-numFeatures: 100,
        logreg_d12da15c7c9d-regParam: 0.1
}, {
        hashingTF_5a96e1199390-numFeatures: 1000,
        logreg_d12da15c7c9d-regParam: 0.1
}, {
        hashingTF_5a96e1199390-numFeatures: 10,
        logreg_d12da15c7c9d-regParam: 0.01
}, {
        hashingTF_5a96e1199390-numFeatures: 100,
        logreg_d12da15c7c9d-regParam: 0.01
}, {
        hashingTF_5a96e1199390-numFeatures: 1000,
```

交叉-验证方法

```scala
// 建立一个交叉验证的评估器，设置评估器的参数
val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new BinaryClassificationEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(2) // Use 3+ in practice

// 运行交叉验证评估器，得到最佳参数集的模型.
val cvModel = cv.fit(training)
```

```
cv: org.apache.spark.ml.tuning.CrossValidator = cv_533362264f35
cvModel: org.apache.spark.ml.tuning.CrossValidatorModel = cv_533362264f35
```

交叉-验证方法

```
// 测试数据.
val test = spark.createDataFrame(Seq(
  (4L, "spark i j k"),
  (5L, "l m n"),
  (6L, "mapreduce spark"),
  (7L, "apache hadoop"))).toDF("id", "text")

// 测试， cvModel会选择最佳lrModel进行预测.
cvModel.transform(test)
  .select("id", "text", "probability", "prediction")
  .collect()
  .foreach {
    case Row(id: Long, text: String, prob: Vector, prediction: Double) =>
      println(s"($id, $text) --> prob=$prob, prediction=$prediction")
  }
```

```
test: org.apache.spark.sql.DataFrame = [id: bigint, text: string]
(4, spark i j k) --> prob=[0.2580684222584646,0.7419315777415353], prediction=1.0
(5, l m n) --> prob=[0.9185597412653915,0.08144025873460856], prediction=0.0
(6, mapreduce spark) --> prob=[0.4320320566391874,0.5679679433608126], prediction=1.0
(7, apache hadoop) --> prob=[0.6766082856652199,0.32339171433478003], prediction=0.0
```

Thanks

FAQ时间