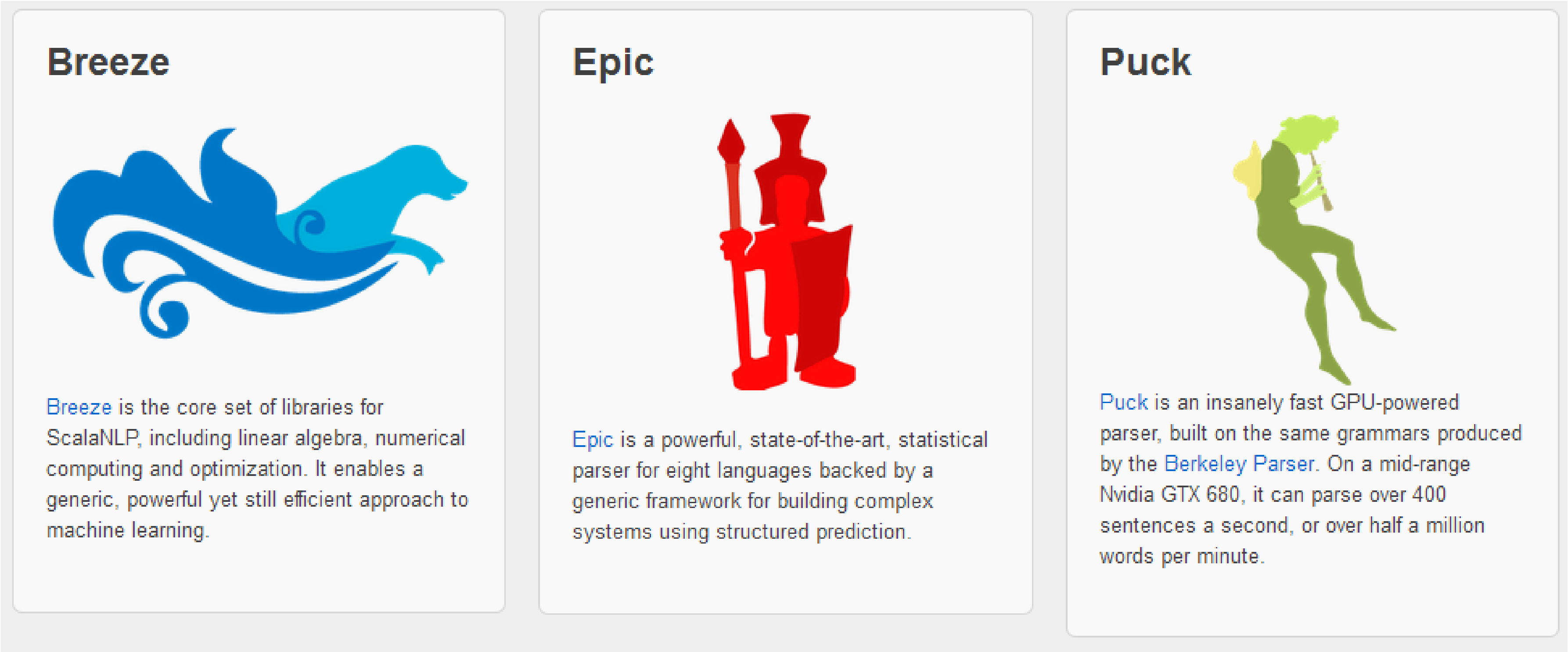


Breeze

Breeze is the core set of libraries for ScalaNLP, including linear algebra, numerical computing and optimization. It enables a generic, powerful yet still efficient approach to machine learning.

Epic

Epic is a powerful, state-of-the-art, statistical parser for eight languages backed by a generic framework for building complex systems using structured prediction.

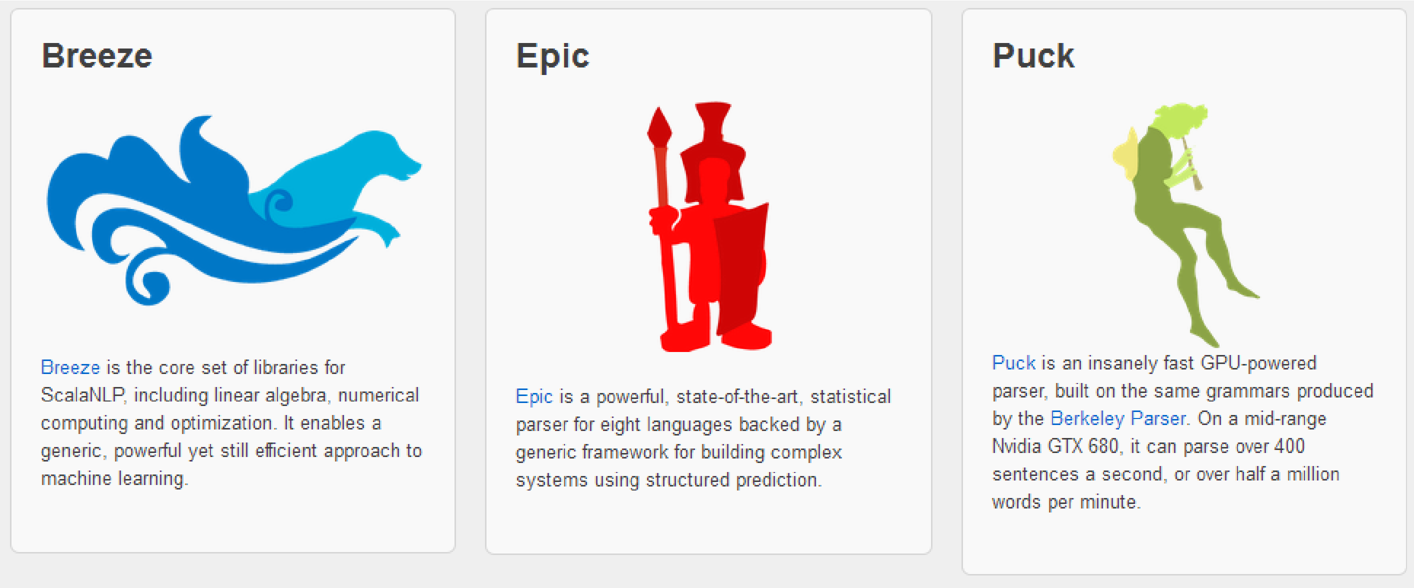
Puck

Puck is an insanely fast GPU-powered parser, built on the same grammars produced by the Berkeley Parser. On a mid-range Nvidia GTX 680, it can parse over 400 sentences a second, or over half a million words per minute.

Spark MLlib机器学习 第2周

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

Breeze is the core set of libraries for ScalaNLP, including linear algebra, numerical computing and optimization. It enables a generic, powerful yet still efficient approach to machine learning.

Epic

Epic is a powerful, state-of-the-art, statistical parser for eight languages backed by a generic framework for building complex systems using structured prediction.

Puck

Puck is an insanely fast GPU-powered parser, built on the same grammars produced by the Berkeley Parser. On a mid-range Nvidia GTX 680, it can parse over 400 sentences a second, or over half a million words per minute.

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

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, 0.174311259949119)
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

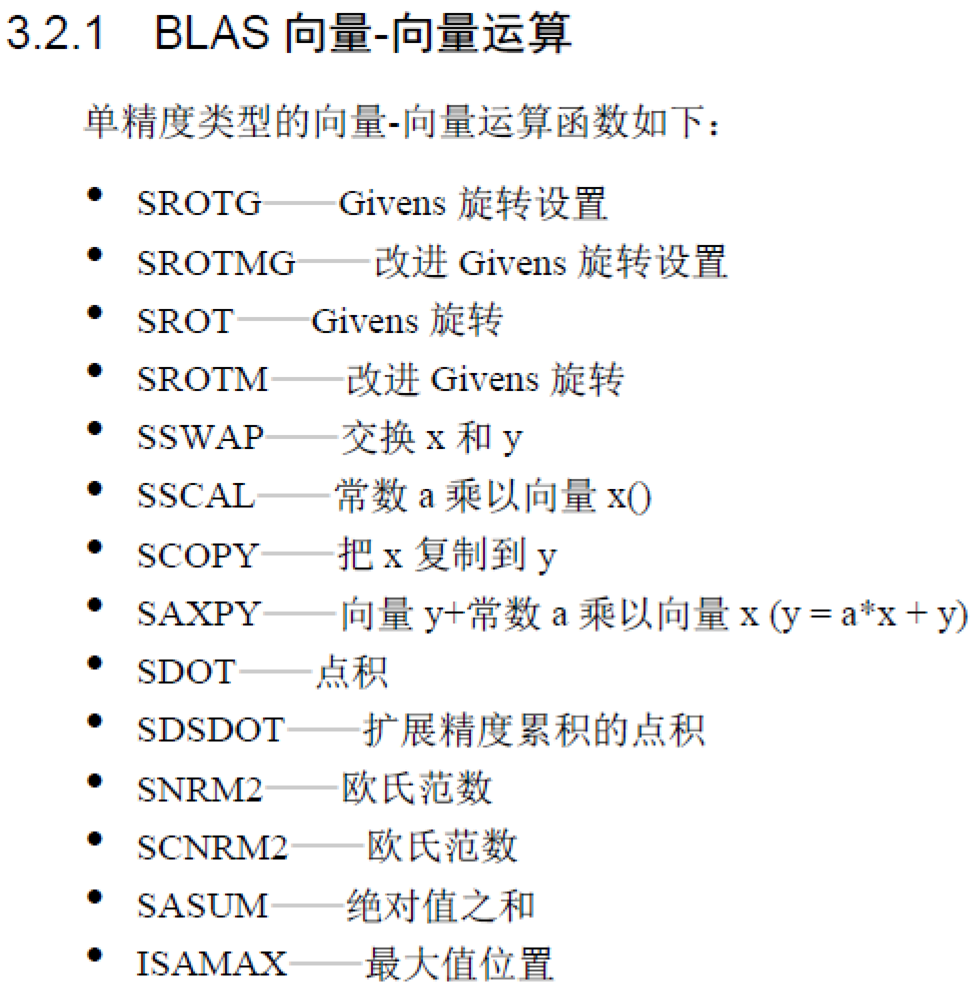
	dim	scalar	vector	vector	scalars	5-element array		prefixes
SUBROUTINE xROTG (					A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(					D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,	C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		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		
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		
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

BLAS介绍

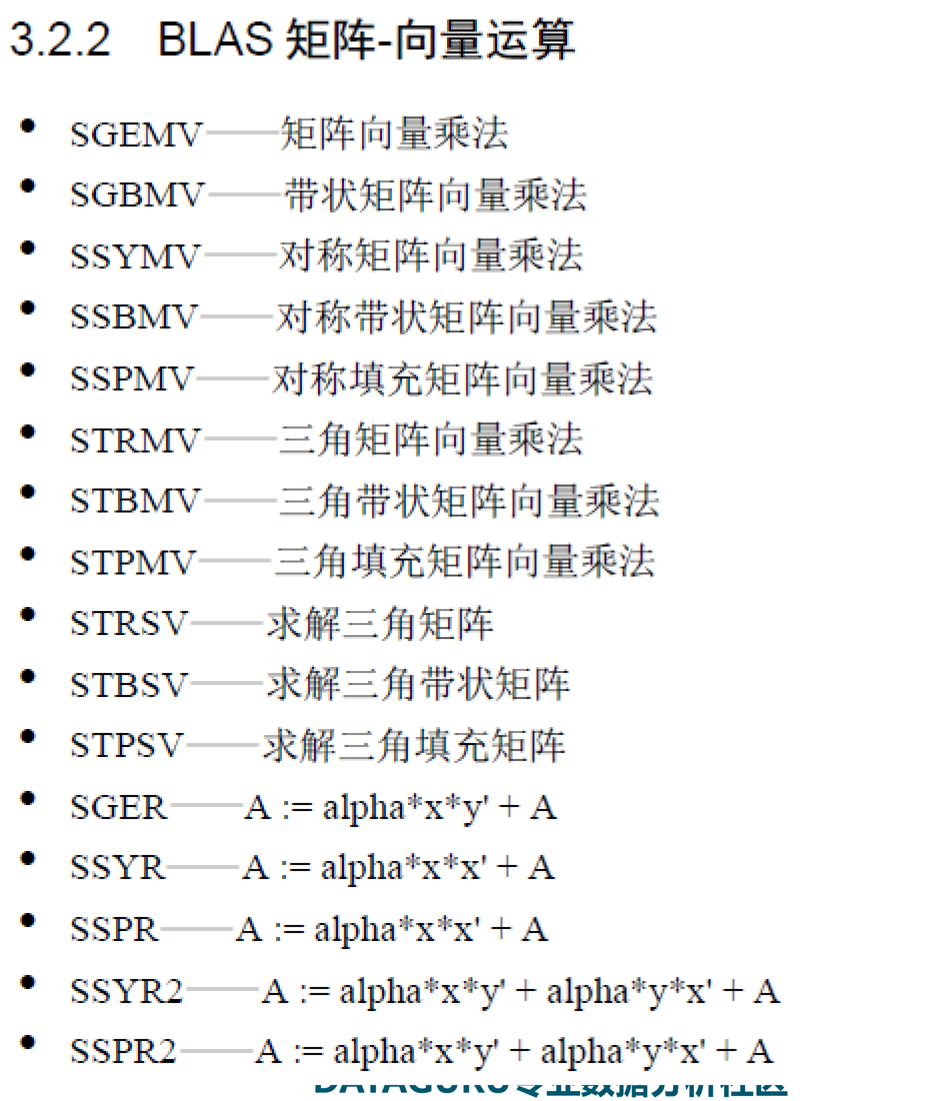


3.2.1 BLAS 向量-向量运算

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

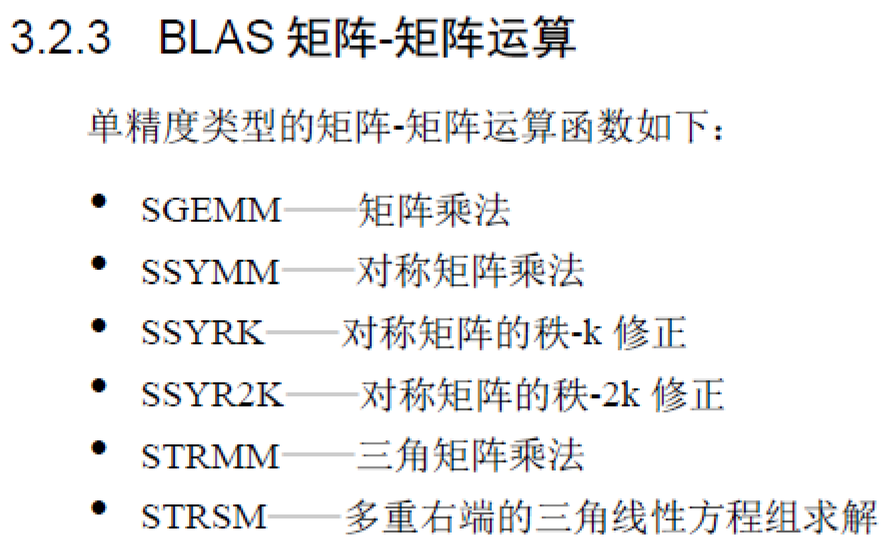
BLAS介绍



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

BLAS介绍



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

法律声明

ATAGURU 炼数成金

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

Thanks

FAQ时间