

- Numpy包

```
1 矩阵:
2 Xm = np.mean(X,axis=0) #axis=0 上下压扁, axis=1左右压扁
3 print("Xm.shape",Xm.shape) #Xm.shape (10,)
4 X1 = X - Xm[None,:] #None表示自适应, 即每行都用这一个行向量
5 print("X1.shape",X1.shape) #X1.shape (442, 10)
6 # Compute the correlations per features
7 syy = np.mean(y1**2)
8 Sxx = np.mean(X1**2,axis=0)
9 Sxy = np.mean(X1*y1[:,None],axis=0)
10
11 # Compute the coefficients and R^2 value per feature
12 beta1 = Sxy/Sxx
13 beta0 = ym - beta1*Xm
14 Rsq = Sxy**2/Sxx/syy #矩阵对应位置的数相乘或除
15 print(Rsq.shape) #(10,)
16
17 #剔除长度为1的轴: 直接改变原数组
18 np.squeeze(logreg.coef_)
19 np.squeeze() #可以直接进行压缩维度, 官方doc的说明如下
20
21 >>> x = np.array([[0], [1], [2]])
22 >>> x.shape
23 (1, 3, 1)
24 >>> np.squeeze(x).shape
25 (3,)
26 >>> np.squeeze(x, axis=(2,)).shape
27 (1, 3)
28 #ravel flatten 也是降到1维数组
29 a1=b.ravel() #改变a1影响b
30 a2=b.flatten() #改变a1不影响b
31
32 #reshape: 第一行然后第二行然后第三行; -1表示自适应; 生成新数据, 不改变a
33 d = a.reshape((-1,1))
34 d=np.reshape(a, (2, 3))
35
36
37 #矩阵合并
38 A = np.column_stack((np.ones(n,), X))
39
40 #生成网格: X和Y同维度, X按x成行扩展, Y按y成列扩展
41 X, Y = np.meshgrid(x,y)
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
1 #生成随机数
2 # (2, 4) 是维度, 两行四列
3 #标准正态分布中随机生成, 期望0标准差1
4 arr1 = np.random.randn(2,4)
5 # [0, 1) 中随机生成
6 arr2 = np.random.rand(2,4)
7 # [0, 1) 中随机生成
8 np.random.uniform(0,1,(nsamp,nx))
9 #随机打乱矩阵
10 numpy.random.shuffle(x) #直接打乱原矩阵
11 numpy.random.permutation(x) #返回一个打乱的副本
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
1 #生成数组
2 numpy.linspace(0,1,10) #start,end,num 左右都闭 endpoint改变是否包含end
3 numpy.arange(0,1,0.1) #start,end stepsize 左闭右开
4 #改变数据类型
5 y0=np.linspace(0,0,10).astype(int)
6 y0.dtype #查看数据类型
```

一、常用属性和函数

.ndim : 维度

.shape : 各维度的尺度 (2, 5)

.size : 元素的个数 10

.dtype : 元素的类型 dtype('int32')

.itemsize : 每个元素的大小, 以字节为单位, 每个元素占4个字节

ndarray数组的创建

np.arange(n) ; 元素从0到n-1的ndarray类型

np.ones(shape): 生成全1

np.zeros(shape, dtype = np.int32) : 生成int32型的全0

np.full(shape, val): 生成全为val

np.eye(n) : 生成单位矩阵 >>> a = np.ones((2,2))

矩阵合并

x1 = np.array([0,0,1,1])

x1.shape=(1,4)

array([[0, 0, 1, 1]])

把行向量变成列向量！！

(4,)代表行向量

>>> b = np.eye(2)

>>> print np.vstack((a,b))

[[1. 1.]

[1. 1.]

[1. 0.]

[0. 1.]]

>>> print np.hstack((a,b)) 注意这里要加两个括号！里面是个tuple

[[1. 1. 1. 0.]

[1. 1. 0. 1.]]

最小二乘拟合，返回第一行是参数

out = np.linalg.lstsq(A,y_tr)

beta = out[0]

np.ones_like(a) : 按数组a的形状生成全1的数组

np.zeros_like(a): 同理

np.full_like (a, val) : 同理

np.linspace (1,10,4) : 根据起止数据等间距地生成数组

np.linspace (1,10,4, endpoint = False) : endpoint 表示10是否作为生成的元素

np.concatenate():

- 数组的维度变换

.reshape(shape) : 不改变当前数组, 依shape生成

.resize(shape) : 改变当前数组, 依shape生成

.swapaxes(ax1, ax2) : 将两个维度调换

.flatten() : 对数组进行降维, 返回折叠后的一位数组

- 数组的类型变换

数据类型的转换 : a.astype(new_type) : eg, a.astype (np.float)

数组向列表的转换 : a.tolist()

数组的索引和切片

- 一维数组切片

a = np.array ([9, 8, 7, 6, 5,])

a[1:4:2] -> array([8, 6]) : a[起始编号: 终止编号 (不含) : 步长]

- 多维数组索引

```
a = np.arange(24).reshape((2, 3, 4))
a[1, 2, 3] 表示 3个维度上的编号， 各个维度的编号用逗号分隔
```

- 多维数组切片

a [:, :, : 2] 缺省时，表示从第0个元素开始，到最后一个元素
数组的运算

imax = np.argmax(Rsq) 返回数组最大值的index

np.abs(a) **np.fabs(a)** : 取各元素的绝对值

np.sqrt(a) : 计算各元素的平方根

np.square(a): 计算各元素的平方

np.log(a) **np.log10(a)** **np.log2(a)** : 计算各元素的自然对数、10、2为底的对数

np.ceil(a) **np.floor(a)** : 计算各元素的ceiling 值, floor值 (ceiling向上取整, floor向下取整)

np rint(a) : 各元素 四舍五入

np.modf(a) : 将数组各元素的小数和整数部分以两个独立数组形式返回

np.exp(a) : 计算各元素的指数值

np.sign(a) : 计算各元素的符号值 1 (+) , 0, -1 (-)

.

np.maximum(a, b) **np.fmax()** : 比较 (或者计算) 元素级的最大值

np.minimum(a, b) **np.fmin()** : 取最小值

np.mod(a, b) : 元素级的模运算

np.copysign(a, b) : 将b中各元素的符号赋值给数组a的对应元素

- numpy随机数函数

numpy 的random子库 **x = np.random.rand(10)** 范围是 (0,1)

numpy.random.uniform(low,high,size) 均匀分布中随机采样

rand(d0, d1, ...,dn) : 各元素是[0, 1) 的浮点数, 服从均匀分布

randn(d0, d1, ...,dn) : 标准正态分布

randint(low, high, (shape) : 依shape创建随机整数或整数数组, 范围是[low, high)

seed(s) : 随机数种子

shuffle(a) : 根据数组a的第一轴进行随机排列, 改变数组a

permutation(a) : 根据数组a的第一轴进行随机排列, 但是不改变原数组, 将生成新数组

choice(a[, size, replace, p]) : 从一维数组a中以概率p抽取元素, 形成size形状新数组, replace表示是否可以重用元素, 默认为False。

- numpy的统计函数

sum(a, axis = None) : 依给定轴axis计算数组a相关元素之和, axis为整数或者元组

mean(a, axis = None) : 同理, 计算平均值

average(a, axis =None, weights=None) : 依给定轴axis计算数组a相关元素的加权平均值

std (a, axis = None) : 同理, 计算标准差

var (a, axis = None) : 计算方差

eg: **np.mean(a, axis =1)** : 对数组a的第二维度的数据进行求平均

a = np.arange(15).reshape(3, 5)

np.average(a, axis =0, weights =[10, 5, 1]) : 对a第一各维度加权求平均, weights中为权重, 注意要和a的第一维匹配

min(a) **max(a)** : 计算数组a的最小值和最大值

argmin(a) **argmax(a)** : 计算数组a的最小、最大值的下标 (注: 是一维的下标)

unravel_index(index, shape) : 根据shape将一维下标index转成多维下标

ptp(a) : 计算数组a最大值和最小值的差

median(a) : 计算数组a中元素的中位数 (中值)

eg: **a = [[15, 14, 13],**

[12, 11, 10]]

np.argmax(a) -> 0

np.unravel_index(np.argmax(a), a.shape) -> (0,0)

*numpy提供的一些常量

```
>>> np.Inf #无穷大
inf
>>> np.NAN #非数字
nan
>>> np.Infinity #无穷大
inf
>>> np.MAXDIMS #数组的最大维度
32
>>> np.NINF #负无穷大
```

```

-inf
>>> np.NaN #非数字
nan
>>> np.NZERO #负0
-0.0
>>> np.pi

```

常用数学函数

NumPy提供常见的如sin,cos和exp

更多函数all, alltrue, any, apply along axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, conjugate, corrcoeff, cov, cross, cumprod, cumsum, diff, dot, floor, inner, inv, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sometrue, sort, std, sum, trace, transpose, var, vdot, vectorize

数组的subset

```

x = np.random.rand(10) # 10 random elements from 0 to 1
x1 = x[2:5]             # Elements 2,3,4 (Note 5 is NOT included)
x2 = x[:4]              # Elements 0,1,2,3 (Starts at 0, element 4 NOT included)
x3 = x[7:]              # Elements 7,8,9 (Element 7 IS included. Ends at 9 NOT 10)
xlast = x[-1]           # The last element

```

二、Array数组生成方法

Type code C Type Minimum size in bytes

'c' character	1
'b' signed integer	1
'B' unsigned integer	1
'u' Unicode character	2
'h' signed integer	2
'H' unsigned integer	2
'i' signed integer	2
'I' unsigned integer	2
'l' signed integer	4
'L' unsigned integer	4
'f' floating point	4
'd' floating point	8

```

1 from array import *
2 myarray=array("l") //表示创建一个integer类型的数组
3 myarray.append(3) //追加元素
4 myarray.pop() //删除最后一个
5 myarray.remove(x) //删除指定的一个x
6 num=myarray[0] //取第一个值
7 myarray.insert(3,10) //3表示下标
8 myarray.reverse() //数组反序

```

Numpy:

NumPy数组是一个多维数组对象，称为ndarray。其由两部分组成：

实际的数据

描述这些数据的元数据

大部分操作仅针对于元数据，而不改变底层实际的数据。

关于NumPy数组有几点必需了解的：

NumPy数组的下标从0开始。

同一个NumPy数组中所有元素的类型必须是相同的。

在详细介绍NumPy数组之前。先详细介绍下NumPy数组的基本属性。NumPy数组的维数称为秩（rank），一维数组的秩为1，二维数组的秩为2，以此类推。在NumPy中，每一个线性的数组称为是一个轴（axes），秩其实是描述轴的数量。比如说，二维数组相当于两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是NumPy中的轴（axes），第一个轴相当于底层数组，第二个轴是底层数组里的数组。而轴的数量——秩，就是数组的维数。

NumPy的数组中比较重要ndarray对象属性有：

1. ndarray.ndim：数组的维数（即数组轴的个数），等于秩。最常见的为二维数组（矩阵）。
2. ndarray.shape：数组的维度。为一个表示数组在每个维度上大小的整数元组。例如二维数组中，表示数组的“行数”和“列数”。ndarray.shape返回一个元组，这个元组的长度就是维度的数目，即ndim属性。
3. ndarray.size：数组元素的总个数，等于shape属性中元组元素的乘积。
4. ndarray.dtype：表示数组中元素类型的对象，可使用标准的Python类型创建或指定dtype。另外也可使用前一篇文章中介绍的NumPy提供的数据类型。
5. ndarray.itemsize：数组中每个元素的字节大小。例如，一个元素类型为float64的数组itemsize属性值为8（float64占用64个bits，每个字节长度为8，所以64/8，占用8个字节），又如，一个元素类型为complex32的数组item属性为4（32/8）。
6. ndarray.data：包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

代码：

```
1 from numpy import *
2
3 a=array([[1,2],[2,3]])
4 a.ndim//数组的维数
5 2
6 a.shape//数组的维度，即行数和列数
7 (2L, 2L)
8 a.dtype//数组中元素的类型
9 dtype('int32')
10 a.size//数组元素的总个数
11 4
```

注意：

使用array函数创建时，参数必须是由方括号括起来的列表，而不能使用多个数值作为参数调用array。

- >>> a = array(1,2,3,4) # 错误
- >>> a = array([1,2,3,4]) # 正确

可以在创建时显式指定数组中元素的类型：

- >>> c = array([[1,2], [3,4]], dtype=complex)
- >>> c
- array([[1.+0.j, 2.+0.j],
- [3.+0.j, 4.+0.j]])

NumPy提供一个类似arange的函数返回一个数列形式的数组：

arange(4)

输出—array([0, 1, 2, 3])

arange(10,30,5)//10-30步长为5

输出—array([10, 15, 20, 25])

x = np.array(range(1,5))**2

当arange使用浮点数参数时，由于浮点数精度有限，通常无法预测获得的元素个数。因此，最好使用函数linspace去接收我们想要的元素个数来代替用range来指定步长。

np.linspace(10,30,5)//这里5表示的是要输出5个数

输出—array([10., 15., 20., 25., 30.]

保证右边界也输出的方法:

```
step = 2.5
x2 = np.arange(0,10+step/2,step)
print(x2)
```

操作可以矢量化 (对数组中每个元素进行操作)

Most operations can be vectorized meaning that the operation is applied to each component.

```
In [16]: x = np.array(range(2,6))
          y = x**2
          print("x = " + str(x))
          print("y = " + str(y))
```