

```

1 #!/usr/bin/env python3
2 # vim: ts=2 sw=2 sts=2 et :
3
4 #
5 #      dew
6 #      ~~~~~\W~~~~~\/~
7 #      ~~~~~\W~~~~~\/~
8 #      O  _/##::/ (O "===_ O, (O  /~
9 #      :::/ (O  "===_ O, (O  /~
10 #      :::/ (O  "===_ O, (O  /~
11 #      :::/ (O  "===_ O, (O  /~
12 #      :::/ (O  "===_ O, (O  /~
13 #      :::/ (O  "===_ O, (O  /~
14 #      :::/ (O  "===_ O, (O  /~
15 #      :::/ (O  "===_ O, (O  /~
16 #      :::/ (O  "===_ O, (O  /~
17 #      :::/ (O  "===_ O, (O  /~
18 #      :::/ (O  "===_ O, (O  /~
19 #      :::/ (O  "===_ O, (O  /~
20 #      :::/ (O  "===_ O, (O  /~
21 #      :::/ (O  "===_ O, (O  /~
22 #      :::/ (O  "===_ O, (O  /~
23 #      :::/ (O  "===_ O, (O  /~
24 #      :::/ (O  "===_ O, (O  /~
25 #      :::/ (O  "===_ O, (O  /~
26 #      :::/ (O  "===_ O, (O  /~
27 #      :::/ (O  "===_ O, (O  /~
28 #      :::/ (O  "===_ O, (O  /~
29 #
30 /sublime.py [OPTIONS]
31 (c)2022 Tim Menzies <timmm@ieee.org> unlicense.org.
32 Sublime's unsupervised bifurcation:
33 let's infer minimal explanations.
34
35 OPTIONS:
36
37 -Max      max numbers to keep      : 512
38 -Some      find 'far' in this many eggs : 512
39 -cautious  On any crash, stop+show stack : False
40 -data      data file                : ./data/aut093.csv
41 -enough    min leaf size            : .5
42 -help      show help                : False
43 -far       how far to look in 'Some' : .9
44 -p         distance coefficient      : 2
45 -seed      random number seed       : 10019
46 -todo      start up task             : nothing
47 -xsmall    Cohen's small effect     : .35
48
49 ## See Also
50
51 [issues](https://github.com/timmm/sublime/issues)
52 âM-^@â [repo](https://github.com/timmm/sublime)
53 âM-^@â [view source](/docs/pdf)
54
55
56 ## Algorithm
57
58 Stochastic clustering to generate tiny models. Uses random projections
59 to divide the space. Then, optionally, explain the clusters by
60 unsupervised iterative dichotomization using ranges that most
61 distinguish sibling clusters.
62
63 e.g.1: just bi-cluster on two distant points
64 ...
65 /sublime.py -c -s $RANDOM -t cluster
66
67 .. : 398
68 .. : 199
69 .. : 99
70 .. : 49 Lbs- Acc+ Mpg+
71 .. : 24 : [2255, 15.5, 30]
72 .. : 25 : [2575, 16.4, 30]
73 .. : 50
74 .. : 25 : [2110, 16.4, 30] <== best
75 .. : 25 : [2205, 16, 30]
76 .. : 100
77 .. : 50
78 .. : 25 : [2234, 15.5, 30]
79 .. : 25 : [2278, 16.5, 30]
80 .. : 50
81 .. : 25 : [2220, 15.5, 30]
82 .. : 25 : [2320, 15.8, 30]
83 .. : 199
84 .. : 99
85 .. : 49
86 .. : 24 : [2451, 16.5, 20]
87 .. : 25 : [3021, 15.5, 20]
88 .. : 50
89 .. : 25 : [3425, 17.6, 20]
90 .. : 25 : [3155, 16.7, 20]
91 .. : 100
92 .. : 50
93 .. : 25 : [4141, 13.5, 10]
94 .. : 25 : [4054, 13.2, 20]
95 .. : 50
96 .. : 25 : [4425, 11, 10]
97 .. : 25 : [4129, 13, 10]
98 ...
99
100 e.g. #2, as above but split on range that mist divides data
101 ...
102 /sublime.py -c -s $RANDOM -t xplain
103 Lbs- Acc+ Mgg+
104 : 398 : [2807, 15.5, 20]
105
106 198 <= Lbs < 454 : 167 : [3725, 14.5, 20]
107 .. Modl < 72 : 34 : [3609, 13, 20]
108 .. Modl >= 72 : 133 : [3735, 14.9, 20]
109
110 .. Cylr < 8 : 56 : [3336, 17, 20]
111 .. : 77 <= Modl < 82 : 22 : [3410, 17.1, 20]
112 .. Modl < 77 or Modl >= 82 : 34 : [3233, 17, 20]
113 .. Cylr >= 8 : 77 : [4129, 13.2, 20]
114 .. Modl < 75 : 37 : [4274, 13, 10]
115 .. Modl >= 75 : 40 : [3962, 13.5, 20]
116 .. Lbs >= 302 : 35 : [4054, 13.2, 20]
117 Lbs < 198 or Lbs >= 454 : 231 : [2290, 16, 30] <== best
118 ...

```

```

119 ## License
120
121 This is free and unencumbered software released into the public
122 domain.
123
124 Anyone is free to copy, modify, publish, use, compile, sell, or
125 distribute this software, either in source code form or as a compiled
126 binary, for any purpose, commercial or non-commercial, and by any
127 means.
128
129 In jurisdictions that recognize copyright laws, the author or authors
130 of this software dedicate any and all copyright interest in the
131 software to the public domain. We make this dedication for the
132 benefit of the public at large and to the detriment of our heirs
133 and successors. We intend this dedication to be an overt act of
134 relinquishment in perpetuity of all present and future rights to
135 this software under copyright law.
136
137 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
138 EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
139 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
140 IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
141 OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
142 ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
143 OR OTHER DEALINGS IN THE SOFTWARE.
144
145 For more information, please refer to <http://unlicense.org/>
146
147
148 import traceback, random, math, sys, re
149 from random import random as r
150 from typing import Any

```

```

151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 def any(a:list) -> Any:
160     "Return a random item."
161     return a[anywhere(a)]
162
163 def anywhere(a:list) -> int:
164     "Return a random index of list 'a'."
165     return random.randint(0, len(a)-1)
166
167 big = sys.maxsize
168
169 def atom(x):
170     "Return a number or trimmed string."
171     x=x.strip()
172     if x=="True": return True
173     elif x=="False": return False
174     else:
175         try: return int(x)
176         except: return float(x)
177         except: return x.strip()
178
179 def demo(do,all):
180     "Maybe run a demo, if we want it, resetting random seed first."
181     todo = dir(all)
182     if do and do != "all":
183         todo = [x for x in dir(all) if x.startswith(do)]
184     for one in todo:
185         fun = all.__dict__[one]
186         if type(fun)==type(demo):
187             random.seed(the.seed)
188             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
189             try:
190                 fun()
191                 print("PASS:", doc)
192             except Exception as e:
193                 all.fails += 1
194                 if the.cautious: traceback.print_exc(); exit(1)
195                 else: print("FAIL:", doc, e)
196             exit(all.fails)
197
198 def file(f):
199     "Iterator. Returns one row at a time, as cells."
200     with open(f) as fp:
201         for line in fp:
202             line = re.sub(r'([\n\r\v\ ]|#.*)', '', line)
203             if line:
204                 yield [atom(cell.strip()) for cell in line.split(",")]
205
206 def first(a:list) -> Any:
207     "Return first item."
208     return a[0]
209
210 def merge(b4:list) -> list:
211     "While we can find similar adjacent things, merge them."
212     j,n,now = -1,len(b4),[]
213     while j < n-1:
214         j += 1
215         a = b4[j]
216         if j < n-2:
217             if merged := a.merge(b4[j+1]):
218                 a = merged
219                 j += 1 # we will continue, after missing one
220             now += [a]
221         # if 'now' is same size as 'b4', look for any other merges.
222     return b4 if len(now)==len(b4) else merge(now)
223
224 class o(object):
225     "Class that can pretty print its slots, with fast inits."
226     def __init__(i, **d): i.__dict__.update(**d)
227     def __repr__(i):
228         pre = i.__class__.__name__ if isinstance(i,o) else ""
229         return pre+str(
230             {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})
231
232 def options(doc:str) -> o:
233     """Convert 'doc' to options dictionary using command line args.
234     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
235     those on the command line means 'flip the default value'; (2) args need only
236     mention the first few of a key (e.g. -s is enough to select for -seed)."""
237     d={}
238     for line in doc.splitlines():
239         if line and line.startswith(" -"):
240             key, _, x = line.strip()[1:].split("#") # get 1st,last word on each line
241             for j,flag in enumerate(sys.argv):
242                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
243                     x= "True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
244             d[key] = atom(x)
245         if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
246     return o(**d)
247
248 def r() -> float:
249     "Return random number 0..1"
250     return random.random()
251
252 def rn(x:float, n=3) -> float:
253     "Round a number to three decimals."
254     return round(x,n)
255
256 def rN(a:list, n=3) -> list:
257     "Round a list of numbers to three decimals."
258     return [rn(x,n=n) for x in a]
259
260 def second(a:list) -> Any:
261     "Return second item."
262     return a[1]
263
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 class Span(o):
278     """Given two 'Sample's and some 'x' range 'lo..hi'.
279     a 'Span' holds often that range appears in each 'Sample'."""
280     def __init__(i,col, lo, hi, ys=None,):
281         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
282
283     def add(i, x:float, y:Any, inc=1) -> None:
284         "y' is a label identifying one 'Sample' or another."
285         i.lo = min(x, i.lo)
286         i.hi = max(x, i.hi)
287         i.ys.add(y,inc)
288
289     def merge(i, j): # -> Span|None
290         "If the merged span is simpler, return that merge."
291         a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
292         if (i.ys.n==0 or j.ys.n==0 or
293             c.div()*.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
294             return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
295
296     def selects(i,row:list) -> bool:
297         "True if the range accepts the row."
298         x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi
299
300 def show(i, positive=True) -> None:
301     "Show the range."
302     txt = i.col.txt
303     if positive:
304         if i.lo == i.hi: return f"[txt] == {i.lo}"
305         elif i.lo == -big: return f"[txt] < {i.hi}"
306         elif i.hi == big: return f"[txt] >= {i.lo}"
307         else: return f"[i.lo] <= [txt] < {i.hi}"
308     else:
309         if i.lo == i.hi: return f"[txt] != {i.lo}"
310         elif i.lo == -big: return f"[txt] >= {i.hi}"
311         elif i.hi == big: return f"[txt] < {i.lo}"
312         else: return f"[txt] < {i.lo} or [txt] >= {i.hi}"
313
314 def support(i) -> float:
315     "Returns 0..1."
316     return i.ys.n / i.col.n
317
318 @staticmethod
319 def sort(spans : list) -> list:
320     "Good spans have large support and low diversity."
321     divs, supports = Num(), Num()
322     sn = lambda s: supports.norm( s.support())
323     dn = lambda s: divs.norm( s.ys.div())
324     f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
325     for s in spans:
326         divs.add( s.ys.div())
327         supports.add(s.support())
328     return sorted(spans, key=f)
329
330 #
331 #
332 #
333 #
334 #
335 #
336 class Col(o):
337     "Summarize columns."
338     def __init__(i,at=0,txt=""):
339         i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "-" in txt else 1)
340
341     def dist(i,x:Any, y:Any) -> float:
342         return 1 if x=="?" and y=="?" else i.dist1(x,y)
343
344 #
345 #
346 #
347 #
348 #
349 #
350 class Sym(Col):
351     "Summarize symbolic columns."
352     def __init__(i,**kw):
353         super().__init__(**kw)
354         i.has, i.mode, i.mode = {}, None, 0
355
356     def add(i, x:str, inc:int=1) -> str:
357         "Update symbol counts in 'has', updating 'mode' as we go."
358         if x != " ":
359             i.n += inc
360             tmp = i.has[x] = inc + i.has.get(x,0)
361             if tmp > i.mode: i.mode, i.mode = tmp, x
362         return x
363
364     def dist(i,x:str, y:str) ->float:
365         "Distance between two symbols."
366         return 0 if x==y else 1
367
368     def div(i):
369         "Return diversity of this distribution (using entropy)."
370         p = lambda x: x / (1E-31 + i.n)
371         return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
372
373     def merge(i, j):
374         "Merge two 'Sym's."
375         k = Sym(at=i.at, txt=i.txt)
376         for x,n in i.has.items(): k.add(x,n)
377         for x,n in j.has.items(): k.add(x,n)
378         return k
379
380     def mid(i):
381         "Return central tendency of this distribution (using mode)."
382         return i.mode
383
384     def spans(i, j, out):
385         """For each symbol in 'i' and 'j', count the
386         number of times we see it on either side."""
387         xys = [(x,"this",n) for x,n in i.has.items()] + [
388             (x,"that",n) for x,n in j.has.items()]
389         one, last = None,None
390         all = []
391         for x,y,n in sorted(xys, key=first):
392             if x != last:
393                 last = x
394                 one = Span(i, x,x)
395                 all += [one]
396             one.add(x,y,n)
397         if len(all) > 1 : out += all

```

```

396 #
397 #
398 #
399 #
400 class Num(Col):
401     "Summarize numeric columns."
402     def __init__(i,**kw):
403         super().__init__(**kw)
404         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
405
406     def add(i,x: float ,inc=1):
407         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
408         if x != "":
409             i.n += inc
410             i.lo = min(x,i.lo)
411             i.hi = max(x,i.hi)
412             if len(i._all) < i.max : i.ok=False; i._all += [x]
413             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
414         return x
415
416     def all(i):
417         "Return '_all', sorted."
418         if not i.ok: i.ok=True; i._all.sort()
419         return i._all
420
421     def dist1(i,x,y):
422         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
423         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
424         else : x,y = i.norm(x), i.norm(y)
425         return abs(x-y)
426
427     def div(i):
428         """"Report the diversity of this distribution (using standard deviation).
429         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
430         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
431         return (i.per(.9) - i.per(.1)) / 2.56
432
433     def merge(i,j):
434         "Return two 'Num's."
435         k = Num(at=i.at, txt=i.txt)
436         for x in i._all: k.add(x)
437         for x in j._all: k.add(x)
438         return k
439
440     def mid(i):
441         "Return central tendency of this distribution (using median)."
442         return i.per(.5)
443
444     def norm(i,x):
445         "Normalize 'x' to the range 0..1."
446         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
447
448     def per(i,p:float=.5) -> float:
449         "Return the p-th ranked item."
450         a = i.all(); return a[ int(p*len(a)) ]
451
452     def spans(i,j, out):
453         """"Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
454         then count the number of times we hit the bin on other side.
455         Then merge similar adjacent bins."""
456         lo = min(i.lo, j.lo)
457         hi = max(i.hi, j.hi)
458         gap = (hi-lo) / (6/the.xsmall)
459         xys = [(x,"this",1) for x in i._all] + [(x,"that",1) for x in j._all]
460         one = Span(i.lo,lo)
461         all = [one]
462         for x,y,n in sorted(xys, key=first):
463             if one.hi - one.lo > gap:
464                 one = Span(i, one.hi,x)
465                 all += [one]
466             one.add(x,y,n)
467         all = merge(all)
468         all[0].lo = -big
469         all[-1].hi = big
470         if len(all) > 1: out += all
471
472 #
473 #
474 #
475 #
476 #
477
478 class Explain(o):
479     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
480     def __init__(i,here):
481         i.here, i.span, i.yes, i.no = here, None, None, None
482
483     def show(i,pre=""):
484         if not pre:
485             tmp = i.here.mid(i.here.y)
486             print(f"{':40} : {len(i.here.rows):5} : {tmp}")
487         if i.yes:
488             s=f"[pre]{i.span.show(True)}"
489             tmp = i.yes.here.mid(i.yes.here.y)
490             print(f"{s:40} : {len(i.yes.here.rows):5} : {tmp}")
491             i.yes.show(pre + "[. ")
492         if i.no:
493             s=f"[pre]{i.span.show(False)}"
494             tmp = i.no.here.mid(i.no.here.y)
495             print(f"{s:40} : {len(i.no.here.rows):5} : {tmp}")
496             i.no.show(pre + "[. ")
497
498 #
499 #
500 #
501 #
502 #
503
504 class Cluster(o):
505     "Tree with 'left','right' samples, broken at median between far points."
506     def __init__(i,here,x=None,y=None,c=None,mid=None):
507         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
508
509     def show(i,pre=""):
510         s = f"[pre:40] : {len(i.here.rows):5}"
511         print(f"{s} if i.left else f'{s} : {i.here.mid(i.here.y))")
512         for kid in [i.left,i.right]:
513             if kid: kid.show(pre + "[. ")
514

```

```

513 #
514 #
515 #
516 #
517 #
518
519 class Sample(o):
520     "Load, then manage, a set of examples."
521     def __init__(i,init=[]):
522         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [],None
523         if str==type(inits): [i.add(row) for row in file(inits)]
524         if list==type(inits): [i.add(row) for row in inits]
525
526     def add(i,a):
527         def col(at,txt):
528             what = Num if txt[0].isupper() else Sym
529             now = what(at=at, txt=txt)
530             where = i.y if "x" in txt or "-" in txt or "!" in txt else i.x
531             if txt[-1] != " ":
532                 where += [now]
533             if "!" in txt: i.klass = now
534             return now
535         #-----
536         if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
537         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
538
539     def clone(i,init=[]):
540         out = Sample()
541         out.add([col.txt for col in i.cols])
542         [out.add(x) for x in inits]
543         return out
544
545     def cluster(i,top=None):
546         """"Split the data using random projections. Find the span that most
547         separates the data. Divide data on that span."""
548         here = Cluster(i)
549         top = top or i
550         if len(i.rows) >= 2*(len(top.rows)**the.enough):
551             left,right,x,y,c,mid = i.half(top)
552             if len(left.rows) < len(i.rows):
553                 here = Cluster(i,x,y,c,mid)
554             here.left = left.cluster(top)
555             here.right = right.cluster(top)
556         return here
557
558     def dist(i,x,y):
559         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
560         return (d/len(i.x)) ** (1/the.p)
561
562     def div(i,cols=None):
563         return [col.div() for col in (cols or i.all)]
564
565     def far(i, x, rows=None):
566         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
567         return tmp[ int(len(tmp)*the.far) ]
568
569     def half(i, top=None):
570         "Using two faraway points 'x,y' break data at median distance."
571         some= i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
572         top = top or i
573         w = any(some)
574         _,x= top.far(w, some)
575         c,y= top.far(x, some)
576         tmp = [r for _,r in sorted([(top.proj(r,x,y,c),r) for r in i.rows],key=first))]
577         mid= len(tmp)//2
578         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
579
580     def mid(i,cols=None):
581         return [col.mid() for col in (cols or i.all)]
582
583     def proj(i,row,x,y,c):
584         "Find the distance of a 'row' on a line between 'x' and 'y'."
585         a = i.dist(row,x)
586         b = i.dist(row,y)
587         return (a**2 + c**2 - b**2) / (2*c)
588
589     def xplain(i,top=None):
590         """"Split the data using random projections. Find the span that most
591         separates the data. Divide data on that span."""
592         here = Explain(i)
593         top = top or i
594         tiny = len(top.rows)**the.enough
595         if len(i.rows) >= 2*tiny:
596             left, right, *_ = i.half(top)
597             spans = []
598             [lcol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
599             if len(spans) > 0:
600                 here.span = Span.sort(spans)[0]
601                 yes, no = i.clone(), i.clone()
602                 [yes if here.span.selects(row) else no).add(row) for row in i.rows]
603                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
604                 if tiny <= len(no.rows ) < len(i.rows): here.no = no.xplain(top=top)
605             return here
606

```

```

600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 class Demos:
618     "Possible start-up actions."
619     fails=0
620     def opt():
621         "show the config"
622         [print(f"{k}>10}={v}") for k,v in the.__dict__.items()]
623
624     def seed():
625         "seed"
626         assert .494 <= r() <= .495
627
628     def num():
629         "check 'Num'."
630         n = Num()
631         for _ in range(100): n.add(r())
632         assert .30 <= n.div() <= .31, "in range"
633
634     def sym():
635         "check 'Sym'."
636         s = Sym()
637         for x in "aaaabbc": s.add(x)
638         assert 1.37 <= s.div() <= 1.38, "entropy"
639         assert 'a' == s.mid(), "mode"
640
641     def rows():
642         "count rows in a file."
643         assert 399 == len([row for row in file(the.data)])
644
645     def sample():
646         "sampling"
647         s = Sample(the.data)
648         assert 398 == len(s.rows), "length of rows"
649         assert 249 == s.x[-1].has[1], "symbol counts"
650
651     def dist():
652         "distance between rows"
653         s = Sample(the.data)
654         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
655
656     def far():
657         "distant items"
658         s = Sample(the.data)
659         for _ in range(32):
660             a,_ = s.far(any(s.rows))
661             assert a>.5, "large?"
662
663     def clone():
664         "cloning"
665         s = Sample(the.data)
666         s1 = s.clone(s.rows)
667         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
668         for k,v in d1.items():
669             assert d2[k] == v, "clone test"
670
671     def half():
672         "divide data in two"
673         s = Sample(the.data); s1,s2,*_ = s.half()
674         print(s1.mid(s1.y))
675         print(s2.mid(s2.y))
676
677     def cluster():
678         "divide data in two"
679         s = Sample(the.data)
680         s.cluster().show(); print("")
681
682     def xplain():
683         "divide data in two"
684         s = Sample(the.data)
685         s.xplain().show(); print("")
686
687 #-----
688 the=options(__doc__)
689 if __name__ == "__main__": demo(the.todo,Demos)
690
691 """
692 all config local to Sample
693 Example class
694 """

```