

```

1 #!/usr/bin/env python3
2 # vim: ts=2 sw=2 sts=2 et :
3
4 #
5 #      dew
6 #      ~~~~~
7 #      O
8 #      |
9 #      |
10 #      |
11 #      |
12 #      |
13 #      |
14 #      |
15 #      |
16 #      |
17 #      |
18 #      |
19 #      |
20 #      |
21 #      |
22 #      |
23 #      |
24 #      |
25 #      |
26 #      |
27 #      |
28 #      |
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) âM-^@% [repo](https://github.com/timmm/sublime) âM-^@% [source](
52 sublime.pdf)
53
54 ## Algorithm
55
56 Stochastic clustering to generate tiny models. Uses random projections
57 to divide the space. Then, optionally, explain the clusters by
58 unsupervised iterative dichotomization using ranges that most
59 distinguish sibling clusters.
60
61 e.g.1: just bi-cluster on two distant points
62
63 ...
64 /sublime.py -c -s $RANDOM -t cluster
65
66 .. : 398
67 .. : 199
68 .. : 99
69 .. : 49 Lbs- Acc+ Mpg+
70 .. : 24 : [2255, 15.5, 30]
71 .. : 25 : [2575, 16.4, 30]
72 .. : 50
73 .. : 25 : [2110, 16.4, 30] <== best
74 .. : 25 : [2205, 16, 30]
75 .. : 100
76 .. : 50
77 .. : 25 : [2234, 15.5, 30]
78 .. : 25 : [2278, 16.5, 30]
79 .. : 50
80 .. : 25 : [2220, 15.5, 30]
81 .. : 25 : [2320, 15.8, 30]
82 .. : 199
83 .. : 99
84 .. : 49
85 .. : 24 : [2451, 16.5, 20]
86 .. : 25 : [3021, 15.5, 20]
87 .. : 50
88 .. : 25 : [3425, 17.6, 20]
89 .. : 25 : [3155, 16.7, 20]
90 .. : 100
91 .. : 50
92 .. : 25 : [4141, 13.5, 10]
93 .. : 25 : [4054, 13.2, 20]
94 .. : 50
95 .. : 25 : [4425, 11, 10]
96 .. : 25 : [4129, 13, 10]
97
98 e.g. #2, as above but split on range that mist divides data
99
100 ...
101 /sublime.py -c -s $RANDOM -t xplain
102
103 .. : 398 Lbs- Acc+ Mgg+
104 .. : [2807, 15.5, 20]
105 198 <= Lbs < 454 : 167 : [3725, 14.5, 20]
106 .. Modl < 72 : 34 : [3609, 13, 20]
107 .. Modl >= 72 : 133 : [3735, 14.9, 20]
108 .. Cylr < 8 : 56 : [3336, 17, 20]
109 .. 77 <= Modl < 82 : 22 : [3410, 17.1, 20]
110 .. Modl < 77 or Modl >= 82 : 34 : [3233, 17, 20]
111 .. Cylr >= 8 : 77 : [4129, 13.2, 20]
112 .. Modl < 75 : 37 : [4274, 13, 10]
113 .. Modl >= 75 : 40 : [3962, 13.5, 20]
114 .. Lbs >= 302 : 35 : [4054, 13.2, 20]
115 Lbs < 198 or Lbs >= 454 : 231 : [2290, 16, 30] <== best
116

```

```

117 ## License
118
119 This is free and unencumbered software released into the public
120 domain.
121
122 Anyone is free to copy, modify, publish, use, compile, sell, or
123 distribute this software, either in source code form or as a compiled
124 binary, for any purpose, commercial or non-commercial, and by any
125 means.
126
127 In jurisdictions that recognize copyright laws, the author or authors
128 of this software dedicate any and all copyright interest in the
129 software to the public domain. We make this dedication for the
130 benefit of the public at large and to the detriment of our heirs
131 and successors. We intend this dedication to be an overt act of
132 relinquishment in perpetuity of all present and future rights to
133 this software under copyright law.
134
135 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
136 EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
137 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
138 IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
139 OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
140 ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
141 OR OTHER DEALINGS IN THE SOFTWARE.
142
143 For more information, please refer to <http://unlicense.org/>
144
145
146 import traceback, random, math, sys, re
147 from random import random as r
148 from typing import Any

```

```

149 #
150 #
151 #
152 #
153 #
154 #
155 #
156
157 def any(a:list) -> Any:
158     "Return a random item."
159     return a[anywhere(a)]
160
161 def anywhere(a:list) -> int:
162     "Return a random index of list 'a'."
163     return random.randint(0, len(a)-1)
164
165 big = sys.maxsize
166
167 def atom(x):
168     "Return a number or trimmed string."
169     x=x.strip()
170     if x=="True": return True
171     elif x=="False": return False
172     else:
173         try: return int(x)
174         except: return float(x)
175     except: return x.strip()
176
177 def demo(do,all):
178     "Maybe run a demo, if we want it, resetting random seed first."
179     todo = dir(all)
180     if do and do != "all":
181         todo = [x for x in dir(all) if x.startswith(do)]
182     for one in todo:
183         fun = all.__dict__[one]
184         if type(fun)==type(demo):
185             random.seed(the.seed)
186             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
187             try:
188                 fun()
189                 print("PASS:", doc)
190             except Exception as e:
191                 all.fails += 1
192                 if the.cautious: traceback.print_exc(); exit(1)
193                 else: print("FAIL:", doc, e)
194             exit(all.fails)
195
196 def file(f):
197     "Iterator. Returns one row at a time, as cells."
198     with open(f) as fp:
199         for line in fp:
200             line = re.sub(r'([\n\r\v\ ]|#.*)', '', line)
201             if line:
202                 yield [atom(cell.strip()) for cell in line.split(",")]
203
204 def first(a:list) -> Any:
205     "Return first item."
206     return a[0]
207
208
209 def merge(b4:list) -> list:
210     "While we can find similar adjacent things, merge them."
211     j,n,now = -1,len(b4),[]
212     while j < n-1:
213         j += 1
214         a = b4[j]
215         if j < n-2:
216             if merged := a.merge(b4[j+1]):
217                 a = merged
218                 j += 1 # we will continue, after missing one
219             now += [a]
220     # if 'now' is same size as 'b4', look for any other merges.
221     return b4 if len(now)==len(b4) else merge(now)
222
223 class o(object):
224     "Class that can pretty print its slots, with fast inits."
225     def __init__(i, **d): i.__dict__.update(**d)
226     def __repr__(i):
227         pre = i.__class__.__name__ if isinstance(i,o) else ""
228         return pre+str(
229             {k: v for k, v in sorted(i.__dict__.items()) if str(k)[0] != "_"})
230
231 def options(doc:str) -> o:
232     """Convert 'doc' to options dictionary using command line args.
233     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
234     those on the command line means 'flip the default value'; (2) args need only
235     mention the first few of a key (e.g. -s is enough to select for -seed)."""
236     d={}
237     for line in doc.splitlines():
238         if line and line.startswith(" -"):
239             key, _, x = line.strip()[1:].split("#") # get 1st,last word on each line
240             for j,flag in enumerate(sys.argv):
241                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
242                     x= "True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
243             d[key] = atom(x)
244     if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
245     return o(**d)
246
247 def r() -> float:
248     "Return random number 0..1"
249     return random.random()
250
251 def rn(x:float, n=3) -> float:
252     "Round a number to three decimals."
253     return round(x,n)
254
255 def rN(a:list, n=3) -> list:
256     "Round a list of numbers to three decimals."
257     return [rn(x,n=n) for x in a]
258
259 def second(a:list) -> Any:
260     "Return second item."
261     return a[1]

```

```

262 #
263 #
264 #
265 #
266 #
267 #
268 #
269
270 #
271 #
272 #
273 #
274
275 class Span(o):
276     """Given two 'Sample's and some 'x' range 'lo..hi'.
277     a 'Span' holds often that range appears in each 'Sample'."""
278     def __init__(i,col, lo, hi, ys=None,):
279         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
280
281     def add(i, x:float, y:Any, inc=1) -> None:
282         "y' is a label identifying one 'Sample' or another."
283         i.lo = min(x, i.lo)
284         i.hi = max(x, i.hi)
285         i.ys.add(y,inc)
286
287     def merge(i, j): # -> Span|None
288         "If the merged span is simpler, return that merge."
289         a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
290         if (i.ys.n==0 or j.ys.n==0 or
291             c.div()*0.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
292             return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
293
294     def selects(i,row:list) -> bool:
295         "True if the range accepts the row."
296         x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi
297
298     def show(i, positive=True) -> None:
299         "Show the range."
300         txt = i.col.txt
301         if positive:
302             if i.lo == i.hi: return f"[txt] == {i.lo}"
303             elif i.lo == -big: return f"[txt] < {i.hi}"
304             elif i.hi == big: return f"[txt] >= {i.lo}"
305             else: return f"[i.lo] <= [txt] < {i.hi}"
306         else:
307             if i.lo == i.hi: return f"[txt] != {i.lo}"
308             elif i.lo == -big: return f"[txt] >= {i.hi}"
309             elif i.hi == big: return f"[txt] < {i.lo}"
310             else: return f"[txt] < {i.lo} or [txt] >= {i.hi}"
311
312     def support(i) -> float:
313         "Returns 0..1."
314         return i.ys.n / i.col.n
315
316 @staticmethod
317 def sort(spans : list) -> list:
318     "Good spans have large support and low diversity."
319     divs, supports = Num(), Num()
320     sn = lambda s: supports.norm( s.support())
321     dn = lambda s: divs.norm( s.ys.div())
322     f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
323     for s in spans:
324         divs.add( s.ys.div())
325         supports.add(s.support())
326     return sorted(spans, key=f)
327
328 #
329 #
330 #
331 #
332
333 class Col(o):
334     "Summarize columns."
335     def __init__(i,at=0,txt=""):
336         i.n,i.at,i.txt,i.w=0,at,txt,(-1 if "-" in txt else 1)
337
338     def dist(i,x:Any, y:Any) -> float:
339         return 1 if x=="?" and y=="?" else i.dist1(x,y)
340
341 #
342 #
343 #
344 #
345
346 class Sym(Col):
347     "Summarize symbolic columns."
348     def __init__(i,**kw):
349         super().__init__(**kw)
350         i.has, i.mode, i.mode = {}, None, 0
351
352     def add(i, x:str, inc:int=1) -> str:
353         "Update symbol counts in 'has', updating 'mode' as we go."
354         if x != " ":
355             i.n += inc
356             tmp = i.has[x] = inc + i.has.get(x,0)
357             if tmp > i.mode: i.mode, i.mode = tmp, x
358         return x
359
360     def dist(i,x:str, y:str) -> float:
361         "Distance between two symbols."
362         return 0 if x==y else 1
363
364     def div(i):
365         "Return diversity of this distribution (using entropy)."
366         p = lambda x: x / (1E-31 + i.n)
367         return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
368
369     def merge(i, j):
370         "Merge two 'Sym's."
371         k = Sym(at=i.at, txt=i.txt)
372         for x,n in i.has.items(): k.add(x,n)
373         for x,n in j.has.items(): k.add(x,n)
374         return k
375
376     def mid(i):
377         "Return central tendency of this distribution (using mode)."
378         return i.mode
379
380     def spans(i, j, out):
381         """For each symbol in 'i' and 'j', count the
382         number of times we see it on either side."""
383         xys = [(x,"this",n) for x,n in i.has.items()] + [
384             (x,"that",n) for x,n in j.has.items()]
385         one, last = None,None
386         all = []
387         for x,y,n in sorted(xys, key=first):
388             if x != last:
389                 last = x
390                 one = Span(i, x,x)
391                 all += [one]
392             one.add(x,y,n)
393         if len(all) > 1 : out += all

```

```

394 #
395 #
396 #
397 #
398 class Num(Col):
399     "Summarize numeric columns."
400     def __init__(i,**kw):
401         super().__init__(**kw)
402         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
403
404     def add(i,x: float ,inc=1):
405         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
406         if x != "":
407             i.n += inc
408             i.lo = min(x,i.lo)
409             i.hi = max(x,i.hi)
410             if len(i._all) < i.max : i.ok=False; i._all += [x]
411             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
412         return x
413
414     def all(i):
415         "Return '_all', sorted."
416         if not i.ok: i.ok=True; i._all.sort()
417         return i._all
418
419     def dist1(i,x,y):
420         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
421         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
422         else : x,y = i.norm(x), i.norm(y)
423         return abs(x-y)
424
425     def div(i):
426         """"Report the diversity of this distribution (using standard deviation).
427         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
428         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
429         return (i.per(.9) - i.per(.1)) / 2.56
430
431     def merge(i,j):
432         "Return two 'Num's."
433         k = Num(at=i.at, txt=i.txt)
434         for x in i._all: k.add(x)
435         for x in j._all: k.add(x)
436         return k
437
438     def mid(i):
439         "Return central tendency of this distribution (using median)."
440         return i.per(.5)
441
442     def norm(i,x):
443         "Normalize 'x' to the range 0..1."
444         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
445
446     def per(i,p:float=.5) -> float:
447         "Return the p-th ranked item."
448         a = i.all(); return a[ int(p*len(a)) ]
449
450     def spans(i,j, out):
451         """"Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
452         then count the number of times we hit the bin on other side.
453         Then merge similar adjacent bins."""
454         lo = min(i.lo, j.lo)
455         hi = max(i.hi, j.hi)
456         gap = (hi-lo) / (6/the.xsmall)
457         xys = [(x,"this",1) for x in i._all] + [(x,"that",1) for x in j._all]
458         one = Span(i.lo,lo)
459         all = [one]
460         for x,y,n in sorted(xys, key=first):
461             if one.hi - one.lo > gap:
462                 one = Span(i, one.hi,x)
463                 all += [one]
464                 one.add(x,y,n)
465             all = merge(all)
466             all[0].lo = -big
467             all[-1].hi = big
468             if len(all) > 1: out += all
469
470 #
471 #
472 #
473 #
474 #
475
476 class Explain(o):
477     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
478     def __init__(i,here):
479         i.here, i.span, i.yes, i.no = here, None, None, None
480
481     def show(i,pre=""):
482         if not pre:
483             tmp = i.here.mid(i.here.y)
484             print(f"{pre[:40]}: {len(i.here.rows):5} : {tmp}")
485         if i.yes:
486             s=f"{pre}{i.span.show(True)}"
487             tmp = i.yes.here.mid(i.yes.here.y)
488             print(f"{s[:40]}: {len(i.yes.here.rows):5} : {tmp}")
489             i.yes.show(pre + "|. ")
490         if i.no:
491             s=f"{pre}{i.span.show(False)}"
492             tmp = i.no.here.mid(i.no.here.y)
493             print(f"{s[:40]}: {len(i.no.here.rows):5} : {tmp}")
494             i.no.show(pre + "|. ")
495
496 #
497 #
498 #
499 #
500 #
501
502 class Cluster(o):
503     "Tree with 'left','right' samples, broken at median between far points."
504     def __init__(i,here,x=None,y=None,c=None,mid=None):
505         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
506
507     def show(i,pre=""):
508         s = f"{pre[:40]}: {len(i.here.rows):5}"
509         print(f"{s}" if i.left else f"{s} : {i.here.mid(i.here.y)}")
510         for kid in [i.left,i.right]:
511             if kid: kid.show(pre + "|. ")

```

```

511 #
512 #
513 #
514 #
515 #
516
517 class Sample(o):
518     "Load, then manage, a set of examples."
519     def __init__(i,init=[]):
520         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
521         if str == type(inits): [i.add(row) for row in file(inits)]
522         if list == type(inits): [i.add(row) for row in inits]
523
524     def add(i,a):
525         def col(at,txt):
526             what = Num if txt[0].isupper() else Sym
527             now = what(at=at, txt=txt)
528             where = i.y if "x" in txt or "-" in txt or "!" in txt else i.x
529             if txt[-1] != " ":
530                 where += [now]
531                 if "!" in txt: i.klass = now
532             return now
533         #-----
534         if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
535         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
536
537     def clone(i,init=[]):
538         out = Sample()
539         out.add([col.txt for col in i.cols])
540         [out.add(x) for x in inits]
541         return out
542
543     def cluster(i,top=None):
544         """"Split the data using random projections. Find the span that most
545         separates the data. Divide data on that span."""
546         here = Cluster(i)
547         top = top or i
548         if len(i.rows) >= 2*(len(top.rows)**the.enough):
549             left,right,x,y,c,mid = i.half(top)
550             if len(left.rows) < len(i.rows):
551                 here = Cluster(i,x,y,c,mid)
552             here.left = left.cluster(top)
553             here.right = right.cluster(top)
554         return here
555
556     def dist(i,x,y):
557         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
558         return (d/len(i.x)) ** (1/the.p)
559
560     def div(i,cols=None):
561         return [col.div() for col in (cols or i.all)]
562
563     def far(i, x, rows=None):
564         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
565         return tmp[ int(len(tmp)*the.far) ]
566
567     def half(i, top=None):
568         "Using two faraway points 'x,y' break data at median distance."
569         some = i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
570         top = top or i
571         w = any(some)
572         _,x= top.far(w, some)
573         c,y= top.far(x, some)
574         tmp = [r for _,r in sorted([(top.proj(r,x,y,c),r)
575                                     for r in i.rows],key=first))]
576         mid= len(tmp)//2
577         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
578
579     def mid(i,cols=None):
580         return [col.mid() for col in (cols or i.all)]
581
582     def proj(i,row,x,y,c):
583         "Find the distance of a 'row' on a line between 'x' and 'y'."
584         a = i.dist(row,x)
585         b = i.dist(row,y)
586         return (a**2 + c**2 - b**2) / (2*c)
587
588     def xplain(i,top=None):
589         """"Split the data using random projections. Find the span that most
590         separates the data. Divide data on that span."""
591         here = Explain(i)
592         top = top or i
593         tiny = len(top.rows)**the.enough
594         if len(i.rows) >= 2*tiny:
595             left, right, *_ = i.half(top)
596             spans = []
597             [icol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
598             if len(spans) > 0:
599                 here.span = Span.sort(spans)[0]
600                 yes, no = i.clone(), i.clone()
601                 [yes if here.span.selects(row) else no].add(row) for row in i.rows]
602                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
603                 if tiny <= len(no.rows ) < len(i.rows): here.no = no.xplain(top=top)
604             return here

```

```

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

```