

```

1 #!/usr/bin/env python3
2 # vim: ts=2 sw=2 sts=2 et :
3
4 #
5 #      dew
6 #
7 #
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)
52 âM-^@e [repo](https://github.com/timmm/sublime)
53 âM-^@e [repo](sublime.pdf)
54 âM-^@e [src](sublime.pdf)
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 ...
66
67 /sublime.py -c -s $RANDOM -t cluster
68
69 .. : 398
70 .. : 199
71 .. : 99
72 .. : 49 Lbs- Acc+ Mpg+
73 .. : 24 : [2255, 15.5, 30]
74 .. : 25 : [2575, 16.4, 30]
75 .. : 50
76 .. : 25 : [2110, 16.4, 30] <== best
77 .. : 25 : [2205, 16, 30]
78 .. : 100
79 .. : 50
80 .. : 25 : [2234, 15.5, 30]
81 .. : 25 : [2278, 16.5, 30]
82 .. : 50
83 .. : 25 : [2220, 15.5, 30]
84 .. : 25 : [2320, 15.8, 30]
85 .. : 199
86 .. : 99
87 .. : 49
88 .. : 24 : [2451, 16.5, 20]
89 .. : 25 : [3021, 15.5, 20]
90 .. : 50
91 .. : 25 : [3425, 17.6, 20]
92 .. : 25 : [3155, 16.7, 20]
93 .. : 100
94 .. : 50
95 .. : 25 : [4141, 13.5, 10]
96 .. : 25 : [4054, 13.2, 20]
97 .. : 50
98 .. : 25 : [4425, 11, 10]
99 .. : 25 : [4129, 13, 10]
100
101 e.g. #2, as above but split on range that mist divides data
102
103 ...
104 /sublime.py -c -s $RANDOM -t explain
105
106 .. : 398 : [2807, 15.5, 20]
107 198 <= Lbs < 454 : 167 : [3725, 14.5, 20]
108 .. Modl < 72 : 34 : [3609, 13, 20]
109 .. Modl >= 72 : 133 : [3735, 14.9, 20]
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 ...

```

```

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

```

```

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

```

```

265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278
279 class Span(o):
280     """Given two 'Sample's and some 'x' range 'lo..hi'.
281     a 'Span' holds often that range appears in each 'Sample'."""
282     def __init__(i,col, lo, hi, ys=None,):
283         i.col, i.lo, i.hi, i.ys = col, lo, hi, ys or Sym()
284
285     def add(i, x:float, y:Any, inc=1) -> None:
286         "y' is a label identifying one 'Sample' or another."
287         i.lo = min(x, i.lo)
288         i.hi = max(x, i.hi)
289         i.ys.add(y,inc)
290
291     def merge(i, j): # -> Span|None
292         "If the merged span is simpler, return that merge."
293         a, b, c = i.ys, j.ys, i.ys.merge(j.ys)
294         if (i.ys.n==0 or j.ys.n==0 or
295             c.div()*.99 <= (a.n*a.div() + b.n*b.div())/(a.n + b.n)):
296             return Span(i.col, min(i.lo,j.lo),max(i.hi,j.hi), ys=c)
297
298     def selects(i,row:list) -> bool:
299         "True if the range accepts the row."
300         x = row[i.col.at]; return x=="?" or i.lo<=x and x<i.hi
301
302     def show(i, positive=True) -> None:
303         "Show the range."
304         txt = i.col.txt
305         if positive:
306             if i.lo == i.hi: return f"[txt] == {i.lo}"
307             elif i.lo == -big: return f"[txt] < {i.hi}"
308             elif i.hi == big: return f"[txt] >= {i.lo}"
309             else: return f"[i.lo] <= [txt] < {i.hi}"
310         else:
311             if i.lo == i.hi: return f"[txt] != {i.lo}"
312             elif i.lo == -big: return f"[txt] >= {i.hi}"
313             elif i.hi == big: return f"[txt] < {i.lo}"
314             else: return f"[txt] < {i.lo} or [txt] >= {i.hi}"
315
316     def support(i) -> float:
317         "Returns 0..1."
318         return i.ys.n / i.col.n
319
320 @staticmethod
321 def sort(spans : list) -> list:
322     "Good spans have large support and low diversity."
323     divs, supports = Num(), Num()
324     sn = lambda s: supports.norm( s.support())
325     dn = lambda s: divs.norm( s.ys.div())
326     f = lambda s: ((1 - sn(s))*2 + dn(s)**2)**.5/2***.5
327     for s in spans:
328         divs.add( s.ys.div())
329         supports.add(s.support())
330     return sorted(spans, key=f)
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 class Sym(Col):
350     "Summarize symbolic columns."
351     def __init__(i,**kw):
352         super().__init__(**kw)
353         i.has, i.mode, i.mode = {}, None, 0
354
355     def add(i, x:str, inc:int=1) -> str:
356         "Update symbol counts in 'has', updating 'mode' as we go."
357         if x != " ":
358             i.n += inc
359             tmp = i.has[x] = inc + i.has.get(x,0)
360             if tmp > i.mode: i.mode, i.mode = tmp, x
361         return x
362
363     def dist(i,x:str, y:str) -> float:
364         "Distance between two symbols."
365         return 0 if x==y else 1
366
367     def div(i):
368         "Return diversity of this distribution (using entropy)."
369         p = lambda x: x / (1E-31 + i.n)
370         return sum( -p(x)*math.log(p(x),2) for x in i.has.values() )
371
372     def merge(i,j):
373         "Merge two 'Sym's."
374         k = Sym(at=i.at, txt=i.txt)
375         for x,n in i.has.items(): k.add(x,n)
376         for x,n in j.has.items(): k.add(x,n)
377         return k
378
379     def mid(i):
380         "Return central tendency of this distribution (using mode)."
381         return i.mode
382
383     def spans(i,j, out):
384         """For each symbol in 'i' and 'j', count the
385         number of times we see it on either side."""
386         xys = [(x,"this",n) for x,n in i.has.items()] + [
387             (x,"that",n) for x,n in j.has.items()]
388         one, last = None,None
389         all = []
390         for x,y,n in sorted(xys, key=first):
391             if x != last:
392                 last = x
393                 one = Span(i, x,x)
394                 all += [one]
395             one.add(x,y,n)
396             if len(all) > 1 : out += all

```

```

397 #
398 #
399 #
400 #
401 class Num(Col):
402     "Summarize numeric columns."
403     def __init__(i,**kw):
404         super().__init__(**kw)
405         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, the.Max, False
406
407     def add(i,x: float ,inc=1):
408         "Reservoir sampler. If '_all' is full, sometimes replace an item at random."
409         if x != "":
410             i.n += inc
411             i.lo = min(x,i.lo)
412             i.hi = max(x,i.hi)
413             if len(i._all) < i.max : i.ok=False; i._all += [x]
414             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
415         return x
416
417     def all(i):
418         "Return '_all', sorted."
419         if not i.ok: i.ok=True; i._all.sort()
420         return i._all
421
422     def dist1(i,x,y):
423         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
424         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
425         else : x,y = i.norm(x), i.norm(y)
426         return abs(x-y)
427
428     def div(i):
429         """"Report the diversity of this distribution (using standard deviation).
430         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
431         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
432         return (i.per(.9) - i.per(.1)) / 2.56
433
434     def merge(i,j):
435         "Return two 'Num's."
436         k = Num(at=i.at, txt=i.txt)
437         for x in i._all: k.add(x)
438         for x in j._all: k.add(x)
439         return k
440
441     def mid(i):
442         "Return central tendency of this distribution (using median)."
443         return i.per(.5)
444
445     def norm(i,x):
446         "Normalize 'x' to the range 0..1."
447         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
448
449     def per(i,p:float=.5) -> float:
450         "Return the p-th ranked item."
451         a = i.all(); return a[ int(p*len(a)) ]
452
453     def spans(i,j, out):
454         """"Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
455         then count the number of times we hit the bin on other side.
456         Then merge similar adjacent bins."""
457         lo = min(i.lo, j.lo)
458         hi = max(i.hi, j.hi)
459         gap = (hi-lo) / (6/the.xsmall)
460         xys = [(x,"this",1) for x in i._all] + [(x,"that",1) for x in j._all]
461         one = Span(i.lo,lo)
462         all = [one]
463         for x,y,n in sorted(xys, key=first):
464             if one.hi - one.lo > gap:
465                 one = Span(i, one.hi,x)
466                 all += [one]
467                 one.add(x,y,n)
468             all = merge(all)
469             all[0].lo = -big
470             all[-1].hi = big
471             if len(all) > 1: out += all
472
473 #
474 #
475 #
476 #
477 #
478 #
479 class Explain(o):
480     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
481     def __init__(i,here):
482         i.here, i.span, i.yes, i.no = here, None, None, None
483
484     def show(i,pre=""):
485         if not pre:
486             tmp = i.here.mid(i.here.y)
487             print(f"[{pre:40}]: {len(i.here.rows):5} : {tmp}")
488         if i.yes:
489             s=f"[pre]{i.span.show(True)}"
490             tmp = i.yes.here.mid(i.yes.here.y)
491             print(f"[{s:40}]: {len(i.yes.here.rows):5} : {tmp}")
492             i.yes.show(pre + "[. ")
493         if i.no:
494             s=f"[pre]{i.span.show(False)}"
495             tmp = i.no.here.mid(i.no.here.y)
496             print(f"[{s:40}]: {len(i.no.here.rows):5} : {tmp}")
497             i.no.show(pre + "[. ")
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 #
515 #
516 #
517 #
518 #
519 #
520 class Sample(o):
521     "Load, then manage, a set of examples."
522     def __init__(i,init=[]):
523         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
524         if str == type(inits): [i.add(row) for row in file(inits)]
525         if list == type(inits): [i.add(row) for row in inits]
526
527     def add(i,a):
528         def col(at,txt):
529             what = Num if txt[0].isupper() else Sym
530             now = what(at=at, txt=txt)
531             where = i.y if "x" in txt or "-" in txt or "!" in txt else i.x
532             if txt[-1] != " ":
533                 where += [now]
534                 if "!" in txt: i.klass = now
535             return now
536
537         #-----
538         if i.cols: i.rows += [[col.add(a[col.at]) for col in i.cols]]
539         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
540
541     def clone(i,init=[]):
542         out = Sample()
543         out.add([col.txt for col in i.cols])
544         [out.add(x) for x in inits]
545         return out
546
547     def cluster(i,top=None):
548         """"Split the data using random projections. Find the span that most
549         separates the data. Divide data on that span."""
550         here = Cluster(i)
551         top = top or i
552         if len(i.rows) >= 2*(len(top.rows)**the.enough):
553             left,right,x,y,c,mid = i.half(top)
554             if len(left.rows) < len(i.rows):
555                 here = Cluster(i,x,y,c,mid)
556                 here.left = left.cluster(top)
557                 here.right = right.cluster(top)
558             return here
559
560     def dist(i,x,y):
561         d = sum( col.dist(x[col.at], y[col.at])**the.p for col in i.x )
562         return (d/len(i.x)) ** (1/the.p)
563
564     def div(i,cols=None):
565         return [col.div() for col in (cols or i.all)]
566
567     def far(i, x, rows=None):
568         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
569         return tmp[ int(len(tmp)*the.far) ]
570
571     def half(i, top=None):
572         "Using two faraway points 'x,y' break data at median distance."
573         some = i.rows if len(i.rows)<the.Some else random.choices(i.rows, k=the.Some)
574         top = top or i
575         w = any(some)
576         _,x= top.far(w, some)
577         c,y= top.far(x, some)
578         tmp = [r for _,r in sorted([(top.proj(r,x,y,c),r) for r in i.rows],key=first))]
579         mid= len(tmp)//2
580         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
581
582     def mid(i,cols=None):
583         return [col.mid() for col in (cols or i.all)]
584
585     def proj(i,row,x,y,c):
586         "Find the distance of a 'row' on a line between 'x' and 'y'."
587         a = i.dist(row,x)
588         b = i.dist(row,y)
589         return (a**2 + c**2 - b**2) / (2*c)
590
591     def xplain(i,top=None):
592         """"Split the data using random projections. Find the span that most
593         separates the data. Divide data on that span."""
594         here = Explain(i)
595         top = top or i
596         tiny = len(top.rows)**the.enough
597         if len(i.rows) >= 2*tiny:
598             left, right, *_ = i.half(top)
599             spans = []
600             [icol.spans(rcol,spans) for lcol,rcol in zip(left.x, right.x)]
601             if len(spans) > 0:
602                 here.span = Span.sort(spans)[0]
603                 yes, no = i.clone(), i.clone()
604                 [yes if here.span.selects(row) else no).add(row) for row in i.rows]
605                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
606                 if tiny <= len(no.rows) < len(i.rows): here.no = no.xplain(top=top)
607             return here
608
609 #

```

```

610 #
611 #
612 #
613 #
614 #
615 #
616 #
617
618 class Demos:
619     "Possible start-up actions."
620     fails=0
621     def opt():
622         "show the config"
623         [print(f"{k}>10}={v}") for k,v in the.__dict__.items()]
624
625     def seed():
626         "seed"
627         assert .494 <= r() <= .495
628
629     def num():
630         "check 'Num'."
631         n = Num()
632         for _ in range(100): n.add(r())
633         assert .30 <= n.div() <= .31, "in range"
634
635     def sym():
636         "check 'Sym'."
637         s = Sym()
638         for x in "aaaabbc": s.add(x)
639         assert 1.37 <= s.div() <= 1.38, "entropy"
640         assert 'a' == s.mid(), "mode"
641
642     def rows():
643         "count rows in a file."
644         assert 399 == len([row for row in file(the.data)])
645
646     def sample():
647         "sampling"
648         s = Sample(the.data)
649         assert 398 == len(s.rows), "length of rows"
650         assert 249 == s.x[-1].has[1], "symbol counts"
651
652     def dist():
653         "distance between rows"
654         s = Sample(the.data)
655         assert .84 <= s.dist(s.rows[1], s.rows[-1]) <= .842
656
657     def far():
658         "distant items"
659         s = Sample(the.data)
660         for _ in range(32):
661             a,_ = s.far(any(s.rows))
662             assert a>.5, "large?"
663
664     def clone():
665         "cloning"
666         s = Sample(the.data)
667         s1 = s.clone(s.rows)
668         d1,d2 = s.x[0].__dict__, s1.x[0].__dict__
669         for k,v in d1.items():
670             assert d2[k] == v, "clone test"
671
672     def half():
673         "divide data in two"
674         s = Sample(the.data); s1,s2,*_ = s.half()
675         print(s1.mid(s1.y))
676         print(s2.mid(s2.y))
677
678     def cluster():
679         "divide data in two"
680         s = Sample(the.data)
681         s.cluster().show(); print("")
682
683     def xplain():
684         "divide data in two"
685         s = Sample(the.data)
686         s.xplain().show(); print("")
687
688 #-----
689 the=options(__doc__)
690 if __name__ == "__main__": demo(the.todo,Demos)
691
692 """
693 all config local to Sample
694 Example class
695 """

```