

```

1 #!/usr/bin/env python3
2 # vim: ts=2 sw=2 sts=2 et :
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 /sublime.py [OPTIONS]
18 (c)2022 Tim Menzies <timm@ieee.org>, BSD license
19 S.U.B.L.I.M.E. =
20 Sublime's unsupervised bifurcation: let's infer minimal explanations.
21
22 OPTIONS:
23
24 -Max      max numbers to keep      : 512
25 -Some     find 'far' in this many egs : 512
26 -cautious On any crash, stop+show stack : False
27 -data     data file                 : data/auto93.csv
28 -enough   min leaf size             : .5
29 -help     show help                 : False
30 -far      how far to look in 'Some'  : .9
31 -p        distance coefficient       : 2
32 -seed     random number seed        : 10019
33 -todo     start up task              : nothing
34 -xsmall   Cohen's small effect      : .35
35
36 ## See Also
37
38 [issues](https://github.com/timm/sublime/issues)
39 :: [repo](https://github.com/timm/sublime)
40 :: [view source](https://github.com/timm/sublime/blob/main/docs/pdf)
41
42 <a href=https://github.com/timm/sublime/actions/workflows/main.yml><img
43 src=https://github.com/timm/sublime/actions/workflows/main.yml/badge.svg></a>
44 
45 
46 
47 (https://doi.org/10.5281/zenodo.5912461)
48 ## Algorithm
49
50 Stochastic clustering to generate tiny models. Uses random projections
51 to divide the space. Then, optionally, explain the clusters by
52 unsupervised iterative dichotomization using ranges that most
53 distinguish sibling clusters.
54
55 ### Example1: just bi-cluster on two distant points
56 ...
57
58 /sublime.py -c -s $RANDOM -t cluster
59
60 .. : 398
61 .. : 199
62 .. : 99
63 .. : 49 Lbs- Acc+ Mpg+
64 .. : 24 : [2255, 15.5, 30]
65 .. : 25 : [2575, 16.4, 30]
66 .. : 50
67 .. : 25 : [2110, 16.4, 30] <== best
68 .. : 25 : [2205, 16, 30]
69 .. : 100
70 .. : 50
71 .. : 25 : [2234, 15.5, 30]
72 .. : 25 : [2278, 16.5, 30]
73 .. : 50
74 .. : 25 : [2220, 15.5, 30]
75 .. : 25 : [2320, 15.8, 30]
76 .. : 199
77 .. : 99
78 .. : 49
79 .. : 24 : [2451, 16.5, 20]
80 .. : 25 : [3021, 15.5, 20]
81 .. : 50
82 .. : 25 : [3425, 17.6, 20]
83 .. : 25 : [3155, 16.7, 20]
84 .. : 100
85 .. : 50
86 .. : 25 : [4141, 13.5, 10]
87 .. : 25 : [4054, 13.2, 20]
88 .. : 50
89 .. : 25 : [4425, 11, 10]
90 .. : 25 : [4129, 13, 10]
91
92 ### Example2: as above but split on range that most divides data
93 ...
94
95 /sublime.py -c -s $RANDOM -t xplain
96 .. Lbs- Acc+ Mgg+
97 .. : 398 : [2807, 15.5, 20]
98 198 <= Lbs < 454 : 167 : [3725, 14.5, 20]
99 .. Modl < 72 : 34 : [3609, 13, 20]
100 .. Modl >= 72 : 133 : [3735, 14.9, 20]
101 .. .. Cylr < 8 : 56 : [3336, 17, 20]
102 .. .. 77 <= Modl < 82 : 22 : [3410, 17.1, 20]
103 .. .. Modl < 77 or Modl >= 82 : 34 : [3233, 17, 20]
104 .. .. Cylr >= 8 : 77 : [4129, 13.2, 20]
105 .. .. Modl < 75 : 37 : [4274, 13, 10]
106 .. .. Modl >= 75 : 40 : [3962, 13.5, 20]
107 .. .. Lbs >= 302 : 35 : [4054, 13.2, 20]
108 Lbs < 198 or Lbs >= 454 : 231 : [2290, 16, 30] <== best
109 ...
110
111 ## License
112
113 Redistribution and use in source and binary forms, with or without
114 modification, are permitted provided that the following conditions are met:
115 1. Redistributions of source code must retain the above copyright notice, this
116 list of conditions and the following disclaimer.
117 2. Redistributions in binary form must reproduce the above copyright notice,
118 this list of conditions and the following disclaimer in the documentation
119 and/or other materials provided with the distribution.
120
121 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
122 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
123 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
124 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
125 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
126 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
127 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
128 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
129 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
130 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
131 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
132
133
134 import traceback, random, copy, math, sys, re
135 from random import random as r
136 from typing import Any

```

```

137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 def any(a:list) -> Any:
146     "Return a random item."
147     return a[anywhere(a)]
148
149 def anywhere(a:list) -> int:
150     "Return a random index of list 'a'."
151     return random.randint(0, len(a)-1)
152
153 big = sys.maxsize
154
155 def atom(x):
156     "Return a number or trimmed string."
157     x=x.strip()
158     if x=="True": return True
159     elif x=="False": return False
160     else:
161         try: return int(x)
162         except: return float(x)
163         except: return x.strip()
164
165 def demo(do,all):
166     "Maybe run a demo, if we want it, resetting random seed first."
167     todo = dir(all)
168     if do and do != "all":
169         todo = [x for x in dir(all) if x.startswith(do)]
170     for one in todo:
171         fun = all.__dict__.get(one,"")
172         if type(fun)==type(demo):
173             random.seed(the.seed)
174             doc = re.sub(r'\n\s+', "\n", fun.__doc__ or "")
175             try:
176                 fun()
177                 print("PASS:", doc)
178             except Exception as e:
179                 all.fails += 0
180                 if the.cautious: traceback.print_exc(); exit(1)
181                 else: print("FAIL:", doc, e)
182             exit(all.fails)
183
184 def file(f):
185     "Iterator. Returns one row at a time, as cells."
186     with open(f) as fp:
187         for line in fp:
188             line = re.sub(r'([\n\r\v\`"]#*)', '', line)
189             if line:
190                 yield [cell.strip() for cell in line.split(",")]
191
192 def first(a:list) -> Any:
193     "Return first item."
194     return a[0]
195
196 def merge(b4:list) -> list:
197     "While we can find similar adjacent things, merge them."
198     j,n,now = -1,len(b4),[]
199     while j < n-1:
200         j += 1
201         a = b4[j]
202         if j < n-2:
203             if merged := a.merge(b4[j+1]):
204                 a = merged
205                 j += 1 # we will continue, after missing one
206             now += [a]
207         # if 'now' is same size as 'b4', look for any other merges.
208     return b4 if len(now)==len(b4) else merge(now)
209
210 class o(object):
211     "Class that can pretty print its slots, with fast inits."
212     def __init__(i, **d): i.__dict__.update(**d)
213     def __repr__(i):
214         pre = i.__class__.__name__ if isinstance(i,o) else ""
215         return pre+'('+'(''.join([f":{k} {v}" for k, v in
216             sorted(i.__dict__.items()) if str(k)[0] != "_"]))+')?'
217
218 def options(doc:str) -> o:
219     """Convert 'doc' to options dictionary using command line args.
220     Args cause two 'shorthands': (1) boolean flags have no arguments (and mentioning
221     those on the command line means 'flip the default value'; (2) args need only
222     mention the first few of a key (e.g. -s is enough to select for -seed)."""
223     d={}
224     for line in doc.splitlines():
225         if line and line.startswith(" -"):
226             key, *_ = line.strip()[1:].split("#") # get 1st,last word on each line
227             for j,flag in enumerate(sys.argv):
228                 if flag and flag[0]=="-" and key.startswith(flag[1:]):
229                     x="True" if x=="False" else "False" if x=="True" else sys.argv[j+1])
230                     d[key] = atom(x)
231             if d["help"]: exit(print(re.sub(r'\n#.*', "", doc, flags=re.S)))
232     return o(**d)
233
234 def r() -> float:
235     "Return random number 0..1"
236     return random.random()
237
238 def rn(x:float, n=3) -> float:
239     "Round a number to three decimals."
240     return round(x,n)
241
242 def rN(a:list, n=3) -> list:
243     "Round a list of numbers to three decimals."
244     return [rn(x,n=n) for x in a]
245
246 def second(a:list) -> Any:
247     "Return second item."
248     return a[1]

```

```

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 class Num(Col):
391     "Summarize numeric columns."
392     def __init__(i,size,**kw):
393         super().__init__(**kw)
394         i._all, i.lo, i.hi, i.max, i.ok = [], 1E32, -1E32, size, False
395
396     def add(i,x:float,inc=1):
397         "Reservoir sampler. If 'all' is full, sometimes replace an item at random."
398         if x != "":
399             i.n += inc
400             i.lo = min(x,i.lo)
401             i.hi = max(x,i.hi)
402             if len(i._all) < i.max : i.ok=False; i._all += [x]
403             elif r() < i.max/i.n: i.ok=False; i._all[anywhere(i._all)] = x
404         return x
405
406     def all(i):
407         "Return 'all'.sorted."
408         if not i.ok: i.ok=True; i._all.sort()
409         return i._all
410
411     def dist1(i,x,y):
412         if x=="?": y=i.norm(y); x=(1 if y<.5 else 0)
413         elif y=="?": x=i.norm(x); y=(1 if x<.5 else 0)
414         else : x,y = i.norm(x), i.norm(y)
415         return abs(x-y)
416
417     def div(i):
418         """Report the diversity of this distribution (using standard deviation).
419         &pm;2, 2.56, 3 &sigma; is 66,90,95%, of the mass. 2&sigma;. So one
420         standard deviation is (90-10)th divide by 2.4 times &sigma;."""
421         return (i.per(.9) - i.per(.1)) / 2.56
422
423     def merge(i,j):
424         "Return two Num's."
425         k = Num(i.max, at=i.at, txt=i.txt)
426         for x in i._all: k.add(x)
427         for x in j._all: k.add(x)
428         return k
429
430     def mid(i):
431         "Return central tendency of this distribution (using median)."
432         return i.per(.5)
433
434     def norm(i,x):
435         "Normalize 'x' to the range 0..1."
436         return 0 if i.hi-i.lo < 1E-9 else (x-i.lo)/(i.hi-i.lo)
437
438     def per(i,p:float=.5) -> float:
439         "Return the p-th ranked item."
440         a = i.all(); return a[ int(p*len(a)) ]
441
442     def prep(i,x):
443         "Return 'x' as a float."
444         return x if x=="?" else float(x)
445
446     def spans(i,j, bins, out):
447         """Divide the whole space 'lo' to 'hi' into, say, 'xsmall'=16 bin,
448         then count the number of times we the bin on other side.
449         Then merge similar adjacent bins."""
450         lo = min(i.lo, j.lo)
451         hi = max(i.hi, j.hi)
452         gap = (hi-lo) / bins
453         xys = [(x,"this",1) for x in i._all] + [
454             (x,"that",1) for x in j._all]
455         one = Span(i.lo,lo)
456         all = [one]
457         for x,y,n in sorted(xys, key=first):
458             if one.hi - one.lo > gap:
459                 one = Span(i, one.hi,x)
460                 all += [one]
461             one.add(x,y,n)
462         all = merge(all)
463         all[0].lo = -big
464         all[-1].hi = big
465         if len(all) > 1: out += all
466
467 #
468 #
469 #
470 #
471 #
472 class Explain(o):
473     "Tree with 'yes','no' branches for samples that do/do not match a 'span'."
474     def __init__(i,here):
475         i.here, i.span, i.yes, i.no = here, None, None, None
476
477     def show(i,pre=""):
478         if not pre:
479             tmp = i.here.mid(i.here.y)
480             print(f"[{pre:40}]: {len(i.here.rows):5} : {tmp}")
481         if i.yes:
482             s=f"[pre]{i.span.show(True)}"
483             tmp = i.yes.here.mid(i.yes.here.y)
484             print(f"[s:40] : {len(i.yes.here.rows):5} : {tmp}")
485             i.yes.show(pre + "|.")
486         if i.no:
487             s=f"[pre]{i.span.show(False)}"
488             tmp = i.no.here.mid(i.no.here.y)
489             print(f"[s:40] : {len(i.no.here.rows):5} : {tmp}")
490             i.no.show(pre + "|.")
491
492 #
493 #
494 #
495 #
496 #
497 class Cluster(o):
498     "Tree with 'left','right' samples, broken at median between far points."
499     def __init__(i,here,x=None,y=None,c=None,mid=None):
500         i.here,i.x,i.y,i.c,i.mid,i.left,i.right = here,x,y,c,mid,None,None
501
502     def show(i,pre=""):
503         s = f"[pre:40] : {len(i.here.rows):5}"
504         print(f"[s] " if i.left else f"[s] : {i.here.mid(i.here.y)})")
505         for kid in [i.left,i.right]:
506             if kid: kid.show(pre + "|.")
507
508 #
509 #
510 #
511 #
512 class Sample(o):
513     "Load, then manage, a set of examples."
514
515     def __init__(i, the, inits=[]):
516         i.the = the
517         i.rows, i.cols, i.x, i.y, i.klass = [], [], [], [], None
518         if str ==type(inits): [i.add(row, True) for row in file(inits)]
519         if list==type(inits): [i.add(row) for row in inits]
520
521
522     def add(i, a, raw=False):
523         pre = lambda a,c: c.prep(a[c.at]) if raw else a[c.at]
524         nump = lambda x : x[0].isupper()
525         skipp = lambda x : x[-1]=="."
526         klassp = lambda x : "I" in x
527         goalp = lambda x : "+" in x or "-" in x or klassp(x)
528         #-----
529         def col(at,txt):
530             now = Num(i.the.Max,at=at,txt=txt) if nump(txt) else Sym(at=at,txt=txt)
531             where = i.y if goalp(txt) else i.x
532             if not skipp(txt):
533                 where += [now]
534                 if klassp(txt): i.klass = now
535             return now
536         #-----
537         if i.cols: i.rows += [[col.add(pre(a,col)) for col in i.cols]]
538         else: i.cols = [col(at,txt) for at,txt in enumerate(a)]
539
540     def clone(i,inits=[]):
541         out = Sample(i.the)
542         out.add([col.txt for col in i.cols])
543         [out.add(x) for x in inits]
544         return out
545
546     def cluster(i,top=None):
547         """Split the data using random projections. Find the span that most
548         separates the data. Divide data on that span."""
549         here = Cluster(i)
550         top = top or i
551         if len(i.rows) >= 2*(len(top.rows)**i.the.enough):
552             left,right,x,y,c,mid = i.half(top)
553             if len(left.rows) < len(i.rows):
554                 here = Cluster(i,x,y,c,mid)
555             here.left = left.cluster(top)
556             here.right = right.cluster(top)
557             return here
558
559     def dist(i,x,y):
560         d = sum( col.dist(x[col.at], y[col.at])**i.the.p for col in i.x )
561         return (d/len(i.x)) ** (1/i.the.p)
562
563     def div(i,cols=None):
564         return [col.div() for col in (cols or i.all)]
565
566     def far(i, x, rows=None):
567         tmp = sorted([(i.dist(x,y),y) for y in (rows or i.rows)],key=first)
568         return tmp[ int(len(tmp)*i.the.far) ]
569
570     def half(i, top=None):
571         "Using two faraway points 'x,y' break data at median distance."
572         some = i.rows if len(i.rows)<i.the.Some else random.choices(i.rows, k=the.Some
573         e)
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)
579             for r in i.rows],key=first))]
580         mid = len(tmp)//2
581         return i.clone(tmp[:mid]), i.clone(tmp[mid:]), x, y, c, tmp[mid]
582
583     def mid(i,cols=None):
584         return [col.mid() for col in (cols or i.all)]
585
586     def proj(i,row,x,y,c):
587         "Find the distance of a 'row' on a line between 'x' and 'y'."
588         a = i.dist(row,x)
589         b = i.dist(row,y)
590         return (a**2 + c**2 - b**2) / (2*c)
591
592     def xplain(i,top=None):
593         """Split the data using random projections. Find the span that most
594         separates the data. Divide data on that span."""
595         here = Explain(i)
596         top = top or i
597         tiny = len(top.rows)**top.the.enough
598         if len(i.rows) >= 2*tiny:
599             left, right,*_ = i.half(top)
600             spans = []
601             [lcol.spans(rcol,6/top.the.xsmall,spans) for lcol,rcol
602                 in zip(left.x, right.x)]
603             if len(spans) > 0:
604                 here.span = Span.sort(spans)[0]
605                 yes, no = i.clone(), i.clone()
606                 [(yes if here.span.selects(row) else no).add(row) for row in i.rows]
607                 if tiny <= len(yes.rows) < len(i.rows): here.yes = yes.xplain(top=top)
608                 if tiny <= len(no.rows) < len(i.rows): here.no = no.xplain(top=top)
609                 return here

```

```

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(the)
623
624     def seed():
625         "seed"
626         assert .494 <= r() <= .495
627
628     def num():
629         "check 'Num'."
630         n = Num(512)
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, the.data)
648         print(the.data, len(s.rows))
649         assert 398 == len(s.rows), "length of rows"
650         assert 249 == s.x[-1].has['l'], "symbol counts"
651
652     def dist():
653         "distance between rows"
654         s = Sample(the, 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, 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, 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, the.data)
675         s1,s2,*_ = s.half()
676         print(s1.mid(s1.y))
677         print(s2.mid(s2.y))
678
679     def cluster():
680         "divide data in two"
681         s = Sample(the, the.data)
682         s.cluster().show(); print("")
683
684     def xplain():
685         "divide data in two"
686         s = Sample(the, the.data);
687         s.xplain().show(); print("")
688
689 #-----
690 the=options(__doc__)
691 if __name__ == "__main__": demo(the.todo,Demos)
692
693 """
694 Example class
695 """

```