


```

1
2
3
4
5
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
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

```



```

(defpackage :tiny (:use (:cl)))
(in-package :tiny)
(macro #<load ('(lib/macros "lib/math" "lib/strings" "lib/lists"
"lib/settings" "lib/structs" "lib/egs" )))

(defvar my (settings "
TINY: semi-supervised multi-objective explanation facility.
(c) 2022 Tim Menzies, BSD-2 clause license

USAGE: lisp eg.lisp [OPTIONS] [ARG]"
' (fpar "f" "how far is distant" " .95)
  (fpar "f" "help file" " ./data/autor93.lisp")
  (help "h" "show help" " nil)
  (keep "k" "items to keep" " 256)
  (k "k" "nb low attributes classes" " 1)
  (m "m" "nb low frequency classes" " 2)
  (p "p" "distance coefficient" " 2)
  (seed "s" "random number seed" " 10019)
  (some "S" "how many" " 512)
  (example "e" "example to run" "ls"))))

(macro #<load ('("col/sample" "col/sym" "col/num" "col/cols" "row/row" "row/data"))

col / row
col / row

(defstruct+ cols
  "Factory for making nums or syms."
  names ; list of column names
  all ; all the generated columns
  x ; just the independent columns
  y ; just the dependent columns
  klass) ; just the class col (if it exists)

(defun make-cols (lst)
  "Upper/lowercase words ==> nums/syms. Kept in 'all' and maybe elsewhere."
  (let (all x y kl (at -1))
    (dolist (str lst (names lst :x :y :klass kl :all (reverse all)))
      (let* ((what (if (upper-case-p (char str 0)) #'make-num #'make-sym))
        (col (funcall what str (incf at))))
        (push col all)
        (unless (eq #\- (charn str))
          (if (member (charn str) '#\! #\~ #\+) (push col y) (push col x))
          (if (eq #\! (charn str)) (setf kl col)))))))

col / num
col / num

(defstruct+ num
  "Summarize numeric columns."
  (txt "") ; column name
  (at 0) ; column position
  (n 0) ; #items seen
  (w 1) ; (1,-1) = (maximize, minimize)
  (lo most-positive-fixnum) ; least seen
  (hi most-negative-fixnum) ; most seen
  (_kept (make-sample))) ; items seen

(defun make-num (&optional (s "") (n 0))
  "Create."
  (%make-num :txt s :at n :w (if (eq #\~ (charn s)) -1 1)))

(defmethod add ((i num) (lst cons))
  "Add a list of items."
  (dolist (x lst i) (add i x))

(defmethod add ((i num) x)
  "Add one thing, skipping 'dont know', updating 'lo,hi' and 'kept'."
  (unless (eq x #\?)
    (with-slots (lo hi i)
      (incf (? i n))
      (add (? i _kept) x)
      (setf lo (min x (? i lo))
        hi (max x (? i hi)))))

(defmethod norm ((i num) x)
  "Map 'x' 0..1 (unless its unknown, unless gap too small."
  (with-slots (lo hi i)
    (cond ((eq x #\?) x)
      ((< (- hi lo) 1E-9) 0)
      (t (/ (- x lo) (- hi lo)))))

(defmethod dist ((i num) x y)
  "Gap between things (0..1). For unknowns, assume max distance."
  (cond ((and (eq #\? x) (eq #\? y))
    (return-from dist 1))
    ((eq #\? x) (setf y (norm i y) x (if (< y .5) 1 0)))
    ((eq #\? y) (setf x (norm i x) y (if (< x .5) 1 0)))
    (t (setf x (norm i x) y (norm i y))))
  (abs (- x y)))

(defmethod mid ((i num))
  "Middle."
  (mid (? i _kept)))

(defmethod div ((i num))
  "Diversity."
  (div (? i _kept)))

(defmethod discretize ((i num) x &optional (bins (? my bins)))
  "Max 'x' to one of 'bins' integers."
  (with-slots (lo hi i)
    (let ((b (/ (- hi lo) bins)))
      (if (= hi lo) 1 (* b (floor (+ .5 (/ x b)))))))

col / sample

```

```

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

```

```

"Keep up to 'max' numbers (after which, replace any old with new)."
(_kept ; where to keep
  (make-array 2 :fill-pointer 0 :adjustable t))
(n 0)
max ; how many to keep
ok) ; nil if items added and list not resorted yet

(defun make-sample (&optional (max (! my kept)))
  "Create."
  (%make-sample :max max))

(defmethod add ((i sample) (x number))
  "Update."
  (incf (? i n))
  (let ((size (length (? i _kept))))
    (cond ((< size (? i max))
      (setf (? i ok) nil)
      (vector-push-extend x (? i _kept))
      ((< (randf) (/ (? i n) (? i max)))
        (setf (? i ok) nil)
        (setf (elt (? i _kept) (randi size)) x))))))

(defmethod per ((i sample) p)
  "Return the pth item from 'kept'."
  (let* ((all (sorted i))
    (n (- (length all))))
    (elt all (max 0 (min n (floor (* p n))))))

(defmethod mid ((i sample))
  "Middle."
  (per i .5))

(defmethod div ((i sample))
  "Diversity."
  (/ (- (per i .9) (per i .1)) 2.58))

(defmethod sorted ((i sample))
  "Return 'kept', sorted."
  (unless (? i ok)
    (sort (? i _kept) #'<)
    (setf (? i ok) t)
    (? i _kept))

col / sym
col / sym

(defstruct+ sym
  "Summarize symbolic columns"
  (txt "") ; column name
  (at 0) ; column position
  (n 0) ; #items seen
  kept) ; symbol counts of the items

(defun make-sym (&optional (s n))
  "Create."
  (%make-sym :txt s :at n))

(defmethod add ((i sym) (lst cons))
  "Add a list of items."
  (dolist (x lst i) (add i x))

(defmethod add ((i sym) x)
  "Add one items, skipping 'dont know', update frequency counts."
  (unless (eq x #\?)
    (incf (? i n))
    (incf (getf x (? i kept)))))

(defmethod add ((i sym) x inc)
  "Bulk add of a symbol 'x', 'inc' times."
  (incf (? i n) inc)
  (incf (getf x (? i kept)) inc))

(defmethod mid ((i sym))
  "Middle."
  (loop for (key . n) in (? i kept) maximizing n return key))

(defmethod div ((i sym))
  "Diversity (entropy)."
  (labels ((fun (p) (* -1 (* p (log p 2)))))
    (loop for (_, n) in (? i kept) sum (fun (/ n (? i n)))))

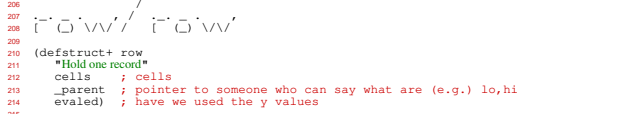
(defmethod dist ((i sym) x y)
  "Gap between 2 items; if unknown, assume max. distance."
  (cond ((and (eq #\? x) (eq #\? y)) 1)
    ((equal x y) 0)
    (t 1)))

```

```

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

```



```

(defstruct+ row
  "Hold one record"
  cells ; cells
  _parent ; pointer to someone who can say what are (e.g.) lo,hi
  evald) ; have we used the y values

(defun make-row (rows lst)
  "Create."
  (%make-row :_parent rows :cells lst))

(defmethod better ((row1 row) (row2 row))
  "Row1 better than row2 if jumping away is better jumping to."
  (let* ((s1 0) (s2 0)
    (cols (cols (? row1 _parent cols y))
      (n (length cols)))
    (setf (? row1 evald) t
      (? row2 evald) t)
    (dolist (col cols (< (/ s1 n) (/ s2 n)))
      (with-slots (at w col)
        (let ((x (norm col (elt (? row1 cells) at)))
          (y (norm col (elt (? row2 cells) at))))
          (decf s1 (exp (* w (/ (- x y) n))))
          (decf s2 (exp (* w (/ (- y x) n))))))))))

(defmethod around ((row1 row) allows)
  "Sort 'allows' by distance to 'row1'."
  (labels ((two (row2) (cons (dist (? row1 _parent row1 row2) row2))
    (sort (mapcar #'two allows) 'car<)))
    (defmethod far ((i row) allows)
      "Return something far away from 'i'. Avoid outliers by only going so 'far'."
      (print 33333)
      (cdr (elt (around i allows)
        (floor (* (length allows) (! my far))))))

col / dist
col / dist

(defstruct+ data
  "Stores multiple rows, and their summaries."
  _rows ; all the rows
  cols) ; summaries of all the columns

(defun make-data (&optional src (i (%make-data)))
  "Eat first row for the column header, add the rest"
  (labels ((top.row.is.special (x) (if (? i cols)
    (push (add i x) (? i _rows))
    (setf (? i cols) (make-cols x))))
    (if (stringp src)
      (with-lines src (lambda (line) (top.row.is.special (cells line))))
      (mapcar #'top.row.is.special src)
      i))

(defmethod clone ((i data) &optional src)
  "Create a new table with same structure as 'i'."
  (make-data (cons (? i cols names) src)))

(defmethod add ((i data) (lst cons))
  "Row creation. Called in we try to add a simple list."
  (add i (make-row i lst)))

(defmethod add ((i data) (row1 row))
  "For all the unskipped columns, update from 'row1'."
  (dolist (cols '(), (? i cols x) (? i cols y)) row1)
  (dolist (col cols)
    (add col (elt (? row1 cells) (? col at)))))

(defmethod dist ((i data) (row1 row) (row2 row))
  "Gap between 'row1', 'row2'. At 'p'=2, this is Euclidean distance."
  (let ((d 0) (n 0) (p (! my p)))
    (dolist (col (? i cols x))
      (incf n)
      (incf d (expt (dist col (elt (? row1 cells) (? col at))
        (elt (? row2 cells) (? col at))) p)))
    (expt (/ d n) (/ 1 p))))

(defmethod half ((i data) &optional (above))
  "Split rows in two by their distance to two remove points."
  (or all (? i _rows)
    (print 1)
    (let (all some left right c tmp)
      (setf all (or all (? i _rows)))
      (setf some (many all (! my some)))
      (print (any some))
      (setf left (or above (far (any some) some)))
      (setf right (far left some))
      (setf c (dist i left right))
      (setf tmp (mapcar (lambda (row)
        (let ((a (dist i row left))
          (b (dist i row right)))
          (cons (/ (+ (* a a) (* c c) (- (* b b))) (* 2 c)) row)))
        all))
      (let ((n 0) lefts rights)
        (dolist (one (sort tmp #'car<))
          (if (< (incf n) (/ (length tmp) 2))
            (push (cdr one) lefts)
            (push (cdr one) rights)))
          (values left right lefts rights c))))

```

```

308 lib/ h n n c i r ' o s
309
310 ; Simple alist access
311 (defmacro ! (l x)
312   "Get into association lists."
313   `(cdr (assoc ',x ,l)))
314
315 (defmacro ? (s x &rest xs)
316   "(? obj x y z) = (slot-value (slot-value (slot-value obj 'x) 'y) 'z)"
317   `(if (null xs) `(slot-value ,s ',x) `(?( slot-value ,s ',x) ,@xs)))
318
319 (defmacro gets (x lst &optional (init 0))
320   "Endure lst has a slot for 'x'. If missing, initialize it with 'init'."
321   `(cdr (or (assoc ,x ,lst :test #'equal)
322             (car (setf ,lst (cons (cons ,x ,init) ,lst))))))
323
324 lib/ h n n t h s
325
326 (defun rnd (number &optional (digits 3))
327   "Round to 'digits' decimal places."
328   (let* ((div (expt 10 digits))
329          (tmp (/ (round (* number div)) div)))
330     (if (zerop digits) (floor tmp) (float tmp)))
331
332 (defvar *seed* 10013)
333 (defun randf (&optional (n 1.0))
334   "Random float in [0,1]"
335   (setf *seed* (mod (* 16807.0d0 *seed* 2147483647.0d0)
336                     (* n - 1.0d0 (/ *seed* 2147483647.0d0))))
337
338 (defun randi (&optional (n 1))
339   "Random int 0..n"
340   (floor (* n (/ (randf 10000000000.0) 10000000000))))
341
342 lib/ s t r i n g s
343
344 (defun charn (x)
345   "Last thing from a string."
346   (and (stringp x)
347        (> (length x) 0)
348        (char x (1- (length x)))))
349
350 (defun trim (x)
351   "Kill leading trailing whitespace."
352   (string-trim '(#\Space #\Tab #\Newline) x))
353
354 (defun thing (x &aux (y (trim x)))
355   "Turn 'x' into a number or string or ??"
356   (cond ((string= y "??") #?)
357         ((string= y "i") t)
358         ((string= y "ml") nil)
359         (t (let ((z (read-from-string y nil nil)))
360              (if (numberp z) z y))))
361
362 (defun splits (str &key (char #\,) (filter #'identity))
363   "Divide 'str' on 'char', filtering all items through 'filter'."
364   (loop for start = 0 then (1+ finish)
365         for finish = (position char str :start start)
366         collecting (funcall filter (trim (subseq str start finish)))
367         until (null finish)))
368
369 ; String to lines or cells of things
370 (defun lines (string) (splits string :char #\Newline))
371 (defun cells (string &key (char #\,)) (splits string :char char :filter #'thing))
372
373 (defun with-lines (file fun)
374   "Call 'fun' for each line in 'file'."
375   (with-open-file (s file)
376     (loop (funcall fun (or (read-line s nil) (return))))))
377
378 lib/ i s t s
379
380 ; sort predicates
381 (defun lt (x) (lambda (a b) (< (slot-value a x) (slot-value b x))))
382 (defun gt (x) (lambda (a b) (> (slot-value a x) (slot-value b x))))
383 (defun car< (a b) (< (car a) (car b)))
384 (defun car> (a b) (> (car a) (car b)))
385
386 ; random sampling (with replacement).
387 (defmethod anv ((l cons)) (any (coerce i 'vector)))
388 (defmethod anv ((l vector)) (elt i (randi (length l))))
389
390 (defmethod manv ((l cons) &optional (n 10)) (many (coerce i 'vector) n))
391 (defmethod many ((l vector) &optional (n 10)) (loop repeat n collect (any i)))
392
393 lib/ s o f t t i n g s
394
395 ; Update 'default' from command line. Boolean flags just flip defaults.
396 (defun cli (key flag help default)
397   "If 'flag' exists on command line, update 'key'."
398   (destructuring-bind (key flag help default) key flag help default
399     (declare (ignore help))
400     (let* ((args #clisp ext:'args*
401              #+sbcl sb-ext:'posix-argv*)
402            (it (member flag args :test 'equalp)))
403       (cons key (cond ((not it) default)
404                     ((equal default t) nil)
405                     ((equal default nil) t)
406                     (t (thing (second it)))))))
407
408 (defun settings (header options)
409   "Update settings. If 'help' is set, print help."
410   (let ((tmp (mapcar (lambda (x) (cli x) options)))
411         (when (! tmp help)
412           (format t "~<-[a-~]-%OPTIONS~%" (lines header))
413           (dolist (one options)
414             (destructuring-bind (flag help default) (cdr one)
415               (format t " ~a -a -a -a-%" flag help default))))
416     tmp))

```

```

423 lib/ s t r u c t s
424
425 (defmacro defstruct* (x doco &body body)
426   "Creates %x for constructor, enables pretty print, hides slots with '._' prefix."
427   (let* ((slots (mapcar (lambda (x) (if (consp x) (car x) x)) body))
428          (show (remove-if (lambda (x) (eq #\_ (char (symbol-name x) 0))) slots)))
429     `(progn
430       (defstruct (.x (:constructor , (intern (format nil "%MAKE--a" x)))) ,@body)
431       (defmethod print-object ((self x) str)
432         (labels ((fun (y) (format nil "~<(-a)-a" y (slot-value self y)))
433                  (format str "-a" (cons '._x (mapcar #'fun ',show)))))))
434
435 lib/ o, c s
436
437 (defvar *egs* nil)
438
439 (defmacro eg (what arg doc &rest src)
440   "define an example"
441   `(push (list ',what ',doc (lambda ,arg ,@src)) *egs*))
442
443 (defun demos (settings all &optional one)
444   "Run 'one' (or 'all') the demos. Reset globals between each run. Return to the operating systems the failure count (so fails=0 means 'success')."
445   (let ((fails 0)
446         (resets (copy-list settings)))
447     (dolist (trio all)
448       (destructuring-bind (what doc fun) trio
449         (setf what (format nil "~<(-a)-" what))
450         (when (member what (list 'all one) :test 'equalp)
451           (loop for (key . value) in resets do
452             (setf (cdr (assoc key settings)) value))
453           (setf *seed* (or (cdr (assoc 'seed settings)) 10019))
454           (unless (eq t (funcall fun))
455             (incf fails)
456             (format t "~&FAIL [-a] -a -%" what doc))))
457       #clisp (ext:exit fails)
458       #sbcl (sb-ext:exit :code fails)))
459
460

```

```

464 o, c
465
466 ; test suite
467 (load "tiny")
468 (in-package :tiny)
469
470 (eg my () "show options" (pprint my) t)
471
472 (eg any () "any, many"
473   (print (sort (loop repeat 20 collect (any # (10 20 30 40))) #'<))
474   (print (sort (many # (10 20 30 40 50 60 70 80 90)
475                     100 110 120 130 140 150) 5) #'<))
476   t)
477
478 (eg sym () "sym"
479   (let ((s (add (make-sym) '(a a a b b c))))
480     (and (= 1.379 (rnd (div s))) (eq 'c (mid s)))))
481
482 (eg sample () "sample"
483   (setf (! my keep) 64)
484   (let ((s (make-sample)))
485     (dotimes (i 100) (add s (1- i)))
486     (and (= 32.170544 (div s)) (= 56 (mid s)))))
487
488 (eg num () "num nums"
489   (setf (! my keep) 64)
490   (let ((n (make-num)))
491     (dotimes (i 100) (add n (1- i)))
492     (and (= 98 (? n hi)) (= 32.170544 (div n)) (= 56 (mid n)))))
493
494 (eg cols () "cols"
495   (print (make-cols '("aa" "bb" "Height" "Weight" "Age-"))))
496   t)
497
498 (eg lines () "lines"
499   (with-lines ".J./data/auto93.csv"
500     (lambda (x) (print (cells x)))))
501   t)
502
503 (eg data () "data"
504   (let ((d (make-data ".J./data/auto93.csv")))
505     (print (? (? d cols) y)))
506   t)
507
508 (eg dist () "dist"
509   (let (all
510         (d (make-data ".J./data/auto93.csv")))
511     (dolist (two (cdr (? d rows)))
512       (push (dist d (car (? d rows)) two) all))
513     (format t "~<[-,3[-]" (sort all #'<)))
514     t))
515
516 (eg half () "half"
517   (let ((d (make-data ".J./data/auto93.csv")))
518     (multiple-value-bind
519       (left right lefts rights c)
520       (half d)
521       (format t "~<-a-%-a-%-a-%" (? left cells) (? right cells) c)))
522   t)
523
524 (demos my *egs* (! my example))

```