brkbad

Conventions

Data

- First row of data are names that describe each column.
- Names ending with `-` or `+` are dependent goals to be minimized or maximized.
- Names ending with `!` are dependent classes.
- Dependent columns are `y` columns (the rest are independent `x` columns).
- Uppercase names are numeric (so the rest are symbolic).
- Names ending with `:` are columns to be skipped.
- Data is read as rows, and stored in a EGS instance.
- Within a EGS, row columns are summarized into NUM or SYM instances.

Inference

- The rows within an EGS are recursive bi-clustered into CLUSTERs using random projections (Fastmap) and Aha's distance metric (that can process numbers and symbols).
- Entropy-based discretization finds BINs that separates each pair of clusters.
- An XPLAIN tree runs the same clustering processing, but data is divided at level using the BIN that most separates the clusters.

Coding

- No globals (so everything is `local`).
- Code 80 characters wide indent with two spaces.

- Format to be read a two-pages-per-page portrait pdf.
- Divide code into section and subsection headings (e.g using figlet)
- Sections are less than 120 lines long (one column in the pdf).
- No lines containing only the word `end` (unless marking the end of a complex for loop or function).
- Usually, if an object contains a list of other objects, that sublist is called `all`.
- If a slot is too big to display, it is declared private (not to be printed) by renaming (e.g.) `slotx` to `_slotx` (so often, `all` becomes `_all`).

Classes

- Spread class code across different sections (so don't overload reader with all details, at one time).
- Show simpler stuff before complex stuff.
- Reserve `i` for `self` (to fit more code per line).
- Don't use inheritance (to simplify readability).
- Use polymorphism (using LUA's delegation trick).

- Define an class of objects with `Thing=class"thing"` and a `function:Thing(args)` creation method.
- Define instances with `new({slot1=value1,slot2=value2,...},Thing)`.
- Instance methods use `.`; e.g. `function Thing.show(i) ... end`.
- Class methods using `:`; e.g. `Thing:new4strings`. Class methods do things like instance creation or manage a set of instances.

Test suites (and demos)

- Define start-up actions as GO functions.

- In GO functions, check for errors with `ok(test,mdf)` (that updates an `fails` counter when not `ok`).
- Define another table called NO so a test can be quickly disabled just by renaming it from `GO.xx` to `NO.xx`.

At top of file

- Trap known globals in `b4`.
- Define all locals at top-of-file (so everyone can access everything).
- Define options in a help string at top of file.

- Define command line options -h (for help); -s (for seeding random numbers) `-t` (for startup actions, so `-t all` means "run everything").

At end of file

- Using `settings`, parse help string to set options, maybe updating from command-line.
- Using `GO.main`, run the actions listed on command line.
- `GO.main` resets random number generator before running an action
- After everything else, look for `rogues` (any global not in `b4`)
- Finally, return the `fails` as the exit status of this code. –]]