```
TOE
   TOE = Timm’s theory of everything
   It aims to  simplify knowledge−level modeling with a little
   data mining.

KNOWLEDGE−LEVEL PROBLEM SOLVING METHODS

   note
      the following text references certain terms that aren’t
      explained till below. So just relax and go with the flow.

   anomaly detector (hmmm... that’s odd)
      − walk through data in "eras" of, say, 100 instances
      − report if median "likelihood(1)" of era i < era[i−1]/2

   verification (do I trust what is going on now?)
      − alert if any app runs on an "era" with anomalies

   classification (give me the executive summary)
      − "likelihood(n)"

   mode identification (what is happening now?)
      − classification using  labels of previous eras
      − if classification is anomalous, declare a new label

   prediction (what will happen now?)
      − classification of this era, then return in the current
        era are the expected values

   planning (how to get there?)
      − find a "contrast set" between a current and goal era.

   control (how to sail upwards)
      − find a "contrast set" between a current era and all
         eras with a higher weight.

   monitor (are we currently smiling?)
      − classification over the utility labels

   explanation
      − contrast set between two eras

   diagnosis (how did we go bad?)
      − explanation, from an eras with a lower to
        a higher utility

   repair (how can we go good?)
      − diagnosis, but flip the weights.
      − also "contrast set" between bad and good,
      − favoring attributes that have the highest frequency
        difference and are cheapest to control

   insert your own here

```

```
FUNCTIONS

   supervised
      count
         − build a frequency table for all
           attribute/range/class  values f[Attr,Range,Class].;
         − e.g. f[sex,male,pregnant]  = 0
         − Note that f[class,label,class] is the
           frequency of class label "Range", which we’ll
           denote f[class] (and "F" is
           the sum of all "f").

      likelihood(1)
         − every instance is labeled "seen"
         − compute likelihood that you have seen this before.
         − prod(f[a,r,"seen"])/f("seen")*(f("seen")/f) = 1)

      likelihood(N)
         − every instance is labeled L
         − compute likelihood that new instance has label L
         − report label with highest likelihood

      contrast
         − given two populations
         − find ranges more frequent in one than the other
         − for top ranked ranges, try with rule generation

   unsupervised
      discretization
         − N bin, equal Fred

      bore (best or rest)
         − discretization on a numeric utility score
         − label top score "best" and the others "rest"

      distance
         − reports distance between two rows

      median
         − propose a node halfway between two others (for discrete
           attributes, move half to the other value, at random)

      GAC
         − builds a tree of nearest pairs
         − if too slow, use sub/micro sampling as a pre−cursor

   sampling
      randomizer
         − randomly re−order rows of the data

      eras
         − spits our data, X instances at a time

      utility
         − add a label to each row based on a scoring function
         − note: simplest one is to just apply the class symbol

      sub−sampling
         − report all rows of the minority class
         − use same number of every other class (at random)

      over−sampling
         − report all rows of the majority class
         − use same number of every other class (repeat at random)

      micro−sampling
         − pick N instances (at random) of all classes
```

```
EXPERIMENT

   hypothesis
      - once the above is working, the building a whole
        range of knowledge-level tasks is a trivial process

   tools
      - we’ll need a generator of data to test this all out

   generator
      sampler(L,P)
         - ascend levels L in the GAC
         - find the average distance of things at level L
           returns random instances within D*L .

      alienator
         - take classified data
         - generates eras of the same class frequency
           as the original data set
         - at interval I, injects a different frequency
           classes at probability P
```