Easier Scripting

Tim Menzies timm@ieee.org

The best way to write less code is to know more about coding. Experienced developers can replace large and complicated code with something much smaller and simpler things. They can do that since they know the tips and tricks.

But where can you learn those tips and tricks? Well, let's try to learn them here. In this document we take something seemingly hard and code it as simple as we can. Along the way, we offer lessons on the tips and tricks used to write the smaller codebase.

Our Case Study: Active Learning

Suppose you are looking at a large number of things, and you do not know which are any good. You could try them all, but that can take a lot of time. In turns out this is a common problem:

- You like fishing but the ocean is a big place so in any day, you can only try a few of your favorite spots.
- You are a software engineer with:
  - too many tests to run;
  - too many Makefile options to try out;
  - a data miner with bewildering number of tuning parameters.

In all these cases, you know the space of possible choices, but you do not have time to score each one. So the task is to score the *fewest things* before finding *some things* that are *good enough* (technically, this is an *active learning* [1] problem for *multi-objective optimization* [2] striving for *heaven* [3]). But what does *good* mean? Well:

- One rule of thumb is that two things are indistinguishable if they differ by less than 35% of the standard deviation [4].
- Another rule of thumb is that, effectively, the standard deviation ranges -3 to 3.
- This means it is the region where we are indistinguishable from best has size $0.35/(3--3) \approx 6\%$ of all.

How hard is it to find 6% of a set of solutions? According to probable correctness theory [5], the certainty C of observing an event with probable p after n random evaluations, is $C = 1-(1-p)^n$. This can be re-arranged to report the number of $n = log(1-C)/log(1-p)$. So at confidence 0.95, we are need $log(1-.95)/log(1-0.06) \approx 50$ random samples.

And we might be able to do better than that. Suppose we have some way to peek at all our examples and guess which are better than the rest. Given a good "peeker", we could, in theory, search our examples via some binary search procedure, using $log_2(50) = 6$ comparisons [6].

This lower value

means that tIf we only want to find something like the best thing, then we only need to get there will be a space of things near the best we don't want to find the best thing, butn

== Main

<ez2.py types>

<1> asdsadasasa asdasd asdasddasas <2> asdasdsdasds

== Conclusion

That's all, folks!

[1] *Active learning* means building a predictive modeling

[2] *Multi-objective optimizers* try to find combinations of

[3] We say that our optimizers are striving to reach some

[4] Standard deviation, or σ, is a measure of much a set of numbers wobble around their middle value. If you sort a 100 numbers in a list a then

[5] Hamlet, Richard G. "Probable correctness theory."

[6] This calculation makes certain optimistic assumption that may not hold in reality. For example, it assumes that our models are simple Gaussians