eLCS

Author: Ryan Urbanowicz
Affiliation: University of Pennsylvania
Email: ryanurb@upenn.edu

eLCS Demo Overview – (eLCS)

Educational Learning Classifier System (eLCS) is a set of learning classifier system (LCS) educational demos designed to introduce students or researchers to the basics of a modern Michigan-style LCS algorithm. This eLCS package includes 5 different implementations of a basic LCS algorithm, as part of a 6 stage set of LCS algorithm code demos. Each eLCS implementation (from demo 1 up to demo 5) progressively add major components of the entire LCS algorithm in order to illustrate how work, how they are coded, and what impact they have on how an LCS algorithm runs. The Demo 5 version of eLCS is most similar to the UCS algorithm described by Bernado-Mansilla. I have used the 6-bit multiplexer as a simple example to be paired with the code, (6Multiplexer_Data_Complete.txt), however I have also included the complete 11-bit multiplexer (11Multiplexer_Data_Complete.txt), and a 2000 instance dataset sampling the 20-bit multiplexer problem (20Multiplexer_Data_2000.txt). Similar to my own ExSTraCS algorithm, eLCS is set up to handle discrete and continuous attributes. This version of eLCS is coded in, and requires Python 3.5 or greater to run. Each of the 5 separate implementations may be run by calling 'eLCS_Run.py'. The configuration files may be edited, to change run parameters, but initially they should be set to run on their own without modification. Users may find it convenient to run and edit eLCS code in Eclipse using PyDev as I have done. Each implementation includes only the minimum code needed to perform the functions they were designed for. This way users can start by examining the simplest version of the code and progress onwards.

This code is intended to be used as an educational tool, or as algorithmic code building blocks. It will be most effective when paired with the LCS introductory textbook but can be used independently as well.

Demo 0 (Rule population)

The purpose of this demo is to familiarize the user with rules and a rule population. I have included, as a rule population example, a real run of the complete (Code Demo 5) eLCS after 5000 iterations (eLCS has learned the problem already with perfect accuracy) (see ExampleRun_eLCS_5000_RulePop.txt). I have also removed some of the rule parameters columns, to keep this example as simple as possible. Users could be directed to examine specific rules, or encouraged to open the rule population in Excel , and try sorting rules by numerosity, accuracy, or initial time stamp in order to examine basic rule properties. We could also just manually select a small set of rules to include as an example rule population for this first break, but it might be good for the reader to be initially exposed to what a complete rule

population might look like. Conditions in the rules included (A_0, A_1, R_0, R_1,R_2, and R_3), making up the multiplexer address (A) and register (R) bits. Class is labeled as Phenotype, since eLCS handles both discrete and continuous endpoints. Also included in the file are the following rule parameters: fitness, accuracy, numerosity, TimeStamp, Initial TimeStamp, and Specificity(just the fraction of specified attributes in a given rule). The rule population is initial ordered by initial Time Stamp, i.e., the iteration in which the rule was originally introduced to the population. There seemed to be little point just loading an existing rule population into an LCS without matching or covering, as nothing really happens. It makes just as much sense to simply start off by examining a rule population. Also included is a short, simple excel file with a small set of example rules.

Code Demo 1 (Code Demo: Matching and Covering)

This first implementation of eLCS is extremely basic. While this each progressive version of eLCS includes the code to load a configuration file, to set run parameters, as well as code to manage an offline environment (load and manage a finite dataset), this first version of eLCS, pretty much only includes the framework of an LCS, i.e. the code to form a population, a match set, a correct set (found in eLCS_ClassifierSet.py), and to construct a classifier (found in eLCS_Classifier.py). Because of their link we introduce both matching and covering together in this initial version. This version can be run from scratch, in which case covering will initial add rules to the population until some random set of covered rules, covers all instances in the dataset (this might be the best way to introduce covering). The code is set to initially run from scratch for 64 iterations (i.e. one cycle through the dataset). Alternatively this version can 'reboot' i.e. load, an existing rule population, so to demonstrate matching more completely. To reboot a population, go into the configuration file and change 'doPopulationReboot' from 0 to 1. In this case, covering will not kick in, since all instances should already be covered. Instead, all matching rules will be displayed. I have encoded this version to use print statements to show what's going on in the algorithm regarding covering and matching. Each iteration, the dataset instance is displayed, followed by any matching rules, as well as any covered rules if covering is activated. The iteration ends with a print out of the iteration number, the current population size and the average rule generality in the population.

Code Demo 2 (Code Demo: Prediction and Rule Population Evaluations)

In this implementation, we have added the prediction array, which allows tracking accuracy of the rule population as an estimate of prediction accuracy, over the last n iterations where n is the tracking frequency (set to the dataset size by default). Rule numerosity is also added to the algorithm at this point, since numerosity plays a role in prediction. However since the only discovery mechanism in the system so far is covering, a classifier numerosity > 1 is only possible, if we load an existing rule population, instead of running the algorithm from scratch. This version is encoded to print statements showing the current instance in the dataset, all rules in the current matchset (including their respective fitnesses) and then the prediction vote for each class, along with the selected prediction. Readers can compare this prediction to the true phenotype of the current instance.

Code Demo 3 (Code Demo: GA Rule Discovery and Parental Selection)

This implementation introduces a panmictic GA for rule discovery including parental selection (tournament or roulette wheel selection are options) , mutation, and uniform crossover. Also we introduce code for two key output files (a print out of the saved rule population, as well as a file with population summary statistics). They are included in this and the following implementations, as it is the first time that eLCS can learn anything interesting enough to be saved and explored. This is also the first time that code for complete rule population evaluations is included. Complete evaluations are included at learning checkpoints, specified in the parameter 'learningIterations' in the configuration file. Note that there is no deletion mechanism yet, so the population size blows up pretty quickly, but the algorithm still works, able to obtain perfect prediction accuracy within 10,000 iterations.

Code Demo 4 (Code Demo: Deletion)

This implementation adds the deletion mechanism, which operates panmicticly as well. This demo shows the key role of deletion in the LCS algorithm, reducing rule population bloat, keeping learning iterations as fast as possible, and the rule population manageable. This version can obtain perfect accuracy on the 6-bit multiplexer problem in under 5,000 iterations.

Code Demo 5 (Code Demo: The Complete Algorithm – Niche GA + Subsumption)

This final version of eLCS puts everything together and adds some additional bells and whistles to get a fully functional eLCS algorithm for supervised learning learning from any kind of dataset. Two key differences include: (1) this version switches to a niche GA (the GA operates in the correct set), as opposed to a panmictic one, and (2) the subsumption mechanism (performs both correct set, and GA based subsumption) has been added to eLCS. Additionally we have added methods for handling balanced accuracy calculations – to accommodate unbalanced, and or multiclass datasets. Also included is a Timer method, which tracks the global run time of the algorithm, as well as the run time used by different major components of the algorithm. eLCS does not include some of the extras from my own ExSTraCS algorithm such as attribute tracking or expert knowledge, as this would probably just be more confusing as part of the text book. This final version can solve the 6-bit multiplexer problem in under 2000 iterations.