Hands-on Workshop on Learning Classifier Systems

Ryan J. Urbanowicz
University of Pennsylvania
Philadelphia, PA, USA
ryanurb@upenn.edu

Will N. Browne
Victoria University of Wellington
Wellington, NZ
will.browne@ecs.vuw.ac.nz

Karthik Kuber
Microsoft
Redmond, WA, USA
karthik.kuber@microsoft.com

ABSTRACT
We will present a hands-on lab to learn the concept and use of Learning Classifier Systems (LCSs)

Keywords: Learning Classifier Systems, Educational LCS, eLCS

1. INTRODUCTION AND SETUP
Welcome to the Educational Learning Classifier System (eLCS). It has the core elements of the functionality that help define the concept of LCSs. It's the same family as the fully featured ExSTraCS [1] system, so it is easy to transfer to a state-of-the-art LCS from this shallow learning curve.

eLCS complements the forthcoming Textbook on Learning Classifier Systems [2]. Each demo is paired with a corresponding section in the textbook. Therefore, there are 5 different versions of an educational learning classifier system (eLCS), as relevant functionality (code) is added to eLCS at each stage. This builds up the eLCS algorithm in its entirety from Demo 1 through to 5. Demo 6 showcases how ExSTraCS may be applied to a real-world data mining example, i.e. large scale bioinformatics. (1) LCS in a Nutshell: Understanding of what an LCS is attempting – how does it classify the training data?, (2) LCS Concepts: Matching and Covering, (3) LCS Functional Cycle: Prediction, Rule Population Evaluations, GA Rule Discovery, Parental Selection and Deletion, (4) LCS Adaptability: Niche GA + Subsumption, (5) LCS Applications: Complete eLCS applied to a complex (toy) problem, (6) ExSTraCS applied to a real-world data mining example.

Example domains are simple initially. The configuration files may be edited, to change run parameters, but initially they should be set to run on their own. All code is in Python version 3.4 or later. Here it is to be run in the Jupyter platform (http://jupyter.org/), as it supports interactive data science. [Note eLCS was originally coded outside of Jupyter in Eclipse using Python 2.7 and Python 3.4 separately, both of which are available for download [3]. Keep in mind that eLCS coded in Python 2.7 requires Python 2+ code will not function in Python 3+. These alternate implementations may be run by calling eLCS Run.py.].

Each demo includes 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.

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

ACM 978-1-4503-4323-7/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2908961.2931738

The demo exercises are to implement several functions in eLCS and view results in a spreadsheet, text file or Python based graphics (preferable). Please set-up Jupyter with Python 3.5 [4]. Please also download eLCS_1.ipynb, … , eLCS_5.ipynb from [3].

2. DEMOS
2.1 LCS in a Nutshell
Firstly, check that the basic eLCS, eLCS_1.ipynb, is running in Jupyter -the "Hello LCS" input code produces an appropriate output when it is run (play button pushed, when input highlighted).

2.1.1 Rule population
The purpose of an LCS is to classify, which it does through a population rules. We have included, as a rule population example, a real run of the complete (Demo 5) eLCS after 5000 iterations (eLCS has learned the problem already with perfect accuracy) (see ExampleRun eLCS 5000 RulePop.txt). We have removed some of the rule parameters columns to keep this example as simple as possible. Users will be directed to examine specific rules, 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. Instead of manually selecting a small set of rules to include as an example rule population for this first Demo, it is good 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 problem address (A) and register (R) bits. Class is labelled as Phenotype, since eLCS handles both discrete and continuous endpoints, which are better generalized as a phenotype. Also included in the file are the following rule parameters: fitness, accuracy, numerosity, TimeStamp, Initial TimeStamp, and Specificity (i.e. the fraction of specified attributes in a given rule). The rule population is initially ordered by initial Time Stamp, i.e., the iteration in which the rule was originally introduced to the population.

2.2 LCS Concepts
2.2.1 Matching and Covering
The first version of eLCS is extremely basic (i.e. Demo 2). While the later versions of eLCS include 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 and to construct a classifier. Because of their link, we introduce both matching and covering together. When run, this version will apply covering to initially add rules to the population until some random set of rules, covers all instances in the dataset. The code is set to initially run for 64 iterations (i.e. one cycle through the dataset).

Alternatively, this version can reboot, i.e. load, an existing rule population, 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 as all instances should already be covered. Instead, all matching rules will be displayed.

We 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.

2.3 LCS Functional Cycle

In an LCS, not all rules match every instance. Not all rules are perfect when created. An accurate, maximally general rule population (the solution) needs to be evolved.

2.3.1 Prediction and Rule Population Evaluations

In the Demo 3 version, 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, if we load an existing rule population, instead of running the algorithm from scratch, only a classifier numerosity of 1 is possible. 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.

2.3.2 GA Rule Discovery and Parental Selection

Demo 3 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.

2.3.3 Deletion

Demo 3 adds the deletion mechanism, which operates panmictically 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.

Further, off-line exercises: 1) Adjust the explore/exploit balance to observe the performance reached and convergence times. 2) Graph, tabulate and visualize the results, including training performance. 3) Interpret and understand the results. 4) Summarize learning in a table, including valid statistical tests if comparing methods. 5) Observe patterns in extracted knowledge. 6) Implement adaptive mutation, then various forms of crossover and then a novel method.

Note: panmictic means unconstrained throughout the population, rather than restricted to local (neighboring) sets.

2.4 LCS Adaptability

2.4.1 The Complete Algorithm: Niche GA + Subsumption

This final version of eLCS (Demo 4) puts everything together and adds some additional bells and whistles to get a fully functional eLCS algorithm for supervised 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. Exercise: compare the difference to where parents of future rules are selected. (2) The subsumption mechanism (performs both correct set, and GA based subsumption) has been added to eLCS. Exercise: explain why certain rules are subsumed and by what classifiers?

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. This final version can solve the 6-bit multiplexer problem in under 2000 iterations.

2.5 LCS Applications

2.5.1 Toy problem using eLCS

This complete version of eLCS is most similar to the UCS algorithm [5]. We have used the complex but toy multiplexer problem: 6Multiplexer Data Complete.txt, however we 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). Exercise: Mess about with the parameter settings!

a. Initially, one at a time. Get a feel for the effect of population size. Reset to a sensible size and then get a feel for a wide range of learning rates. Each time noting the classification performance trend as well as the final achieved performance.

b. Combinations of parameters. Parameters that strongly interacting include: Population Size and P# (particularly for sparse problems such as the parity problem), Population size and genetic algorithm rate (see if you can get the covering loop problem to occur), Learning rate and error threshold ε0.

2.6 Real problem using ExSTraCS

A quick introduction to the ExSTraCS algorithm in contrast with the complete eLCS algorithm will be given. This will be followed with a discussion and demonstration of how this more advanced algorithm can be applied to a real world genetic analysis to find unique complex patterns. More information on ExSTraCS at [1,6].

3. REFERENCES

[1] R. Urbanowicz, J. Moore: ExSTraCS 2.0: description and evaluation of a scalable learning classifier system. Evo. Intell. 8(2) (2015) 89-116

[2] W. Browne, R. Urbanowicz: An Introductory Textbook on Learning Classifier Systems. Springer. In Preparation

[3] Download eLCS: https://github.com/ryanurbs

[4] Download Python: https://www.python.org/downloads/

[5] E. Bernadó-Mansilla, J. Garrell-Guiu: Accuracy-based LCS: models, analysis and applications to classification tasks. Evo. Comp.11(3) (2003) 209-238

[6] http://www.ryanurbanowicz.com/exstracs