CITS4404-G1 Documentation

Release 1.0

Scott McCormack

Nov 03, 2017

CONTENTS:

CHAPTER

ONE

ELCS PACKAGE

1.1 Submodules

1.2 eLCS.Algorithm module

class eLCS.Algorithm.**Algorithm**
Bases: object

The major controlling module of eLCS.

Includes the major run loop which controls learning over a specified number of iterations. Also includes periodic tracking of estimated performance, and checkpoints where complete evaluations of the eLCS rule population is performed.

Two options are available for the initialisation of the algorithm 1. Do a Population reboot using an existing saved rule population, or 2. Build the Population from scratch from given data

doContPopEvaluation(*isTrain*)
Performs evaluation of population via the copied environment.

Specifically developed for continuous phenotype evaulation. The population is maintained unchanging throughout the evaluation. Works on both training and testing data.

Parameters **isTrain** –

Returns

doPopEvaluation(*isTrain*)
Performs a complete evaluation of the current rule population.

The population is unchanged throughout this evaluation. Works on both training and testing data.

Parameters **isTrain** –

Returns

getRuntimeParams()

plotResult()
Plot the runtime params from the execution of the LCS

populationReboot()
Manages the reformation of a previously saved eLCS classifier population

runIteration(*state_phenotype*, *exploreIter*)
Run a single eLCS learning iteration.

Parameters

- **state_phenotype** (*list*) – Listing consisting of the training state and training phenotype
- **exploreIter** (*int*) – The current iteration

Returns

run_eLCS()
Runs eLCS algorithm, runs by default after the class has been initialised

1.3 eLCS.ClassAccuracy module

class eLCS.ClassAccuracy.**ClassAccuracy**
Bases: object

Manages the logistical aspects of balance accuracy calculations.

These can handle unbalanced datasets, and/or datasets with multiple discrete classes.

reportClassAccuracy()
Print to standard out, summary on the class accuracy

updateAccuracy(*thisIsMe*, *accurateClass*)
Increment the appropriate cell of the confusion matrix

Parameters

- **thisIsMe** –
- **accurateClass** –

1.4 eLCS.Classifier module

class eLCS.Classifier.**Classifier**(*a=None*, *b=None*, *c=None*, *d=None*)
Bases: object

This module defines an individual classifier within the rule population, along with all respective parameters.

Also included are classifier-level methods, including constructors(covering, copy, reboot) matching, subsumption, crossover, and mutation. Parameter update methods are also included

Mutation(*state*, *phenotype*)
Mutates the condition of the classifier. Also handles phenotype mutation. This is a niche mutation, which means that the resulting classifier will still match the current instance.

buildMatch(*attRef*, *state*)
Builds a matching condition for the classifierCovering method.

classifierCopy(*clOld*, *exploreIter*)
Constructs an identical Classifier. However, the experience of the copy is set to 0 and the numerosity is set to 1 since this is indeed a new individual in a population. Used by the genetic algorithm to generate offspring based on parent classifiers.

classifierCovering(*setSize*, *exploreIter*, *state*, *phenotype*)
Makes a new classifier when the covering mechanism is triggered.

The new classifier will match the current training instance. Covering will NOT produce a default rule (i.e. a rule with a completely general condition).

The classifier constructs phenotypes for: 1. Discrete Phenotypes 2. Continous Phenotypes

Parameters

- **setSize** (*int*) – The set numerosity sum
- **exploreIter** (*int*) – The current iteration
- **state** (*list*) – The state
- **phenotype** (*int*) – The state's phenotype

continuousPhenotypeMutation(*phenotype*)
Mutate this rule's continuous phenotype.

discretePhenotypeMutation()
Mutate this rule's discrete phenotype.

equals(*cl*)
Returns if the two classifiers are identical in condition and phenotype. This works for discrete or continuous attributes or phenotypes.

getDelProp(*meanFitness*)
Returns the vote for deletion of the classifier.

isMoreGeneral(*cl*)
Returns if the classifier (self) is more general than cl. Check that all attributes specified in self are also specified in cl.

isSubsumer()
Returns if the classifier (self) is a possible subsumer. A classifier must be as or more accurate than the classifier it is trying to subsume.

match(*state*)
Returns if the classifier matches in the current situation.

phenotypeCrossover(*cl*)
Crossover a continuous phenotype

printClassifier()
Formats and returns an output string describing this classifier.

rebootClassifier(*classifierList*)
Rebuilds a saved classifier as part of the population Reboot

setAccuracy(*acc*)
Sets the accuracy of the classifier

setFitness(*fit*)
Sets the fitness of the classifier.

subsumes(*cl*)
Returns if the classifier (self) subsumes cl

uniformCrossover(*cl*)
Applies uniform crossover and returns if the classifiers changed. Handles both discrete and continuous attributes. #SWARTZ: self. is where for the better attributes are more likely to be specified #DEVITO: cl. is where less useful attribute are more likely to be specified

updateAccuracy()
Update the accuracy tracker

updateCorrect()
Increases the correct phenotype tracking by one. Once an epoch has completed, rule accuracy can't change.

updateExperience()
Increases the experience of the classifier by one. Once an epoch has completed, rule accuracy can't change.

updateFitness()
Update the fitness parameter.

updateMatchSetSize(*matchSetSize*)
Updates the average match set size.

updateNumerosity(*num*)
Updates the numberosity of the classifier. Notice that 'num' can be negative!

updateTimeStamp(*ts*)
Sets the time stamp of the classifier.

1.5 eLCS.ClassifierSet module

class eLCS.ClassifierSet.**ClassifierSet**(*pop_reboot_path=None*)
Bases: object

This module handles all the classifier sets

This includes the population, match set and correct sets along with mechanisms and heuristics that act on these sets.

This class can be initialized with the: 1. Creation of a new population, or 2. Reboots the population (i.e. read in from a previously saved population)

addClassifierToPopulation(*cl*, *covering*)
Adds a classifier to the set and increases the microPopSize value accordingly.

Parameters

- **cl** –
- **covering** –

Returns

clearSets()
Clears out references in the match and correct sets for the next learning iteration.

deleteFromCorrectSet(*deleteRef*)
Delete reference to classifier in population, contained in self.corectSet.

Parameters **deleteRef** –

Returns

deleteFromMatchSet(*deleteRef*)
Delete reference to classifier in population, contained in self.matchSet.

Parameters **deleteRef** –

Returns

deleteFromPopulation()
Deletes one classifier in the population.

The classifier that will be deleted is chosen by roulette wheel selection considering the deletion vote. Returns the macro-classifier which got decreased by one micro-classifier.

deletion(*exploreIter*)
Returns the population size back to the maximum set by the user by deleting rules.

Parameters **exploreIter** –

Returns

doCorrectSetSubsumption()
Executes correct set subsumption.

The correct set subsumption looks for the most general subsumer classifier in the correct set and subsumes all classifiers that are more specific than the selected one.

getFitnessSum(*setList*)
Returns the sum of the fitnesses of all classifiers in the set.

Parameters **setList** –

Returns

getIdenticalClassifier(*newCl*)
Looks for an identical classifier in the population.

Parameters **newCl** –

Returns

getIterStampAverage()
Returns the average of the time stamps in the correct set.

getPopFitnessSum()
Returns the sum of the fitnesses of all classifiers in the set.

getPopTrack(*accuracy*, *exploreIter*, *trackingFrequency*)
Returns a formatted output string to be printed to the Learn Track output file.

Parameters

- **accuracy** –
- **exploreIter** –
- **trackingFrequency** –

Returns

insertDiscoveredClassifiers(*cl1*, *cl2*, *clP1*, *clP2*, *exploreIter*)
Inserts both discovered classifiers and activates GA subsumption if turned on.

Also checks for default rule (i.e. rule with completely general condition) and prevents such rules from being added to the population, as it offers no predictive value within eLCS.

Parameters

- **cl1** –
- **cl2** –
- **clP1** –
- **clP2** –
- **exploreIter** –

Returns

makeCorrectSet(*phenotype*)
Constructs a correct set out of the given match set

Parameters **phenotype** –

Returns

makeEvalMatchSet(*state*)
Constructs a match set for evaluation purposes which does not activate either covering or deletion.

Parameters **state** –

Returns

makeMatchSet(*state_phenotype*, *exploreIter*)
Constructs a match set from the population

Covering is initiated if the match set is empty or a rule with the current correct phenotype is absent.

Parameters

- **state_phenotype** (*list*) – Listing consisting of the training state and training phenotype
- **exploreIter** (*int*) – The current iteration

makePop()
Initializes the rule population, as an empty list

rebootPop(*pop_reboot_path*)
Remakes a previously evolved population from a saved text file

Parameters **pop_reboot_path** –

Returns

removeMacroClassifier(*ref*)
Removes the specified (macro-) classifier from the population.

Parameters **ref** –

Returns

runAttGeneralitySum(*isEvaluationSummary*)
Determine the population-wide frequency of attribute specification, and accuracy weighted specification.

Used in complete rule population evaluations.

Parameters **isEvaluationSummary** –

Returns

runGA(*exploreIter*, *state*, *phenotype*)
The genetic discovery mechanism in eLCS is controlled here.

Parameters

- **exploreIter** –
- **state** –
- **phenotype** –

Returns

runPopAveEval(*exploreIter*)
Calculates some summary evaluations across the rule population including average generality.

Parameters **exploreIter** –

Returns

selectClassifierRW()
Selects parents using roulette wheel selection according to the fitness of the classifiers.

Returns

selectClassifierT()
Selects parents using tournament selection according to the fitness of the classifiers.

setIterStamps(*exploreIter*)
Sets the time stamp of all classifiers in the set to the current time.

The current time is the number of exploration steps executed so far.

Parameters **exploreIter** –

Returns

subsumeClassifier(*cl=None*, *cl1P=None*, *cl2P=None*)
Tries to subsume a classifier in the parents.

If no subsumption is possible it tries to subsume it in the current set.

Parameters

- **cl** –
- **cl1P** –
- **cl2P** –

Returns

subsumeClassifier2(*cl*)
Tries to subsume a classifier in the correct set.

If no subsumption is possible the classifier is simply added to the population considering the possibility that there exists an identical classifier.

Parameters **cl** –

Returns

updateSets(*exploreIter*)
Updates all relevant parameters in the current match and correct sets.

Parameters **exploreIter** –

Returns

1.6 eLCS.Constants module

class eLCS.Constants.**Constants**
Bases: object

Stores and manages all algorithm run parameters

Parameters are accessible anywhere in the rest of the algorithm code by importing *cons*

loadParameters(*config_file*)
Load the environment parameters from yaml configuration file

Parameters **config_file** (*str*) – Path to the configuration yaml file

Returns Parameters read from yaml file

Return type dict

parseIterations()
Parse the 'learningIterations' string

Identify the maximum number of learning iterations as well as evaluation checkpoints

referenceEnv(*env*)
Store reference to *OfflineEnvironment* object

Parameters **env** (*Environment*) – An *OfflineEnvironment* file

referenceTimer(*timer*)
Store reference to the Timer object

Parameters **timer** – A timer object

setConstants(*config_file*, *dataset_path*)
Parse the configuration file and save them as global constants

Parameters

- **config_file** (*str*) – Path to the configuration yaml file
- **dataset_path** (*str*) – Directory to the datasets

1.7 eLCS.DataManagement module

class eLCS.DataManagement.**DataManagement**(*trainFile*, *testFile*, *infoList=None*)
Bases: object

Able to manage both training and testing data.

This module loads the dataset, detects and characterizes all attributes in the dataset, handles missing data, and finally formats the data so that it may be conveniently utilized by eLCS.

characterizeAttributes(*rawData*)
Determine range (if continuous) or states (if discrete) for each attribute and saves this information

Parameters **rawData** –

Returns

characterizeDataset(*rawTrainData*)
Detect basic dataset parameters

Parameters **rawTrainData** –

Returns

characterizePhenotype(*rawData*)
Determine range of phenotype values.

Parameters **rawData** –

Returns

compareDataset(*rawTestData*)
Ensures that the attributes in the testing data match those in the training data.

Also stores some information about the testing data.

Parameters **rawTestData** –

Returns

discriminateAttributes(*rawData*)
Determine whether attributes in dataset are discrete or continuous and saves this information.

Parameters **rawData** –

Returns

discriminateClasses(*rawData*)
Determines number of classes and their identifiers.

Only used if phenotype is discrete.

Parameters **rawData** –

Returns

discriminatePhenotype(*rawData*)
Determine whether the phenotype is Discrete(class-based) or Continuous

Parameters **rawData** (*list*) – The raw data file loaded from

formatData(*rawData*)
Get the data into a format convenient for the algorithm to interact with.

Specifically each instance is stored in a list as follows; [Attribute States, Phenotype, InstanceID]

Parameters **rawData** –

Returns

loadData(*dataFile*, *doTrain*)
Load the data file.

Parameters

- **dataFile** –
- **doTrain** –

Returns

1.8 eLCS.OfflineEnvironment module

class eLCS.OfflineEnvironment.**OfflineEnvironment**
Bases: object

In the context of data mining and classification tasks, the environment is a data set with a limited number of instances with X attributes and some endpoint (typically a discrete phenotype or class) of interest.

This module loads the data set, automatically detects features of the data by executing the DataManagement module

getTestInstance()
Returns the current training instance.

getTrainInstance()
Returns the current training instance

newInstance(*isTraining*)
Shifts the environment to the next instance in the data.

Parameters **isTraining** –

Returns

resetDataRef(*isTraining*)
Resets the environment back to the first instance in the current data set.

Parameters **isTraining** –

Returns

startEvaluationMode()
Turns on evaluation mode. Saves the instance we left off in the training data.

stopEvaluationMode()
Turns off evaluation mode. Re-establishes place in dataset.

1.9 eLCS.OutputFileManager module

class eLCS.OutputFileManager.**OutputFileManager**
Bases: object

This module contains the methods for generating the different output files generated by eLCS.

These files are generated at each learning checkpoint, and the last iteration. These include... * writePopStats: Summary of the population statistics * writePop: Outputs a snapshot of the entire rule population including classifier conditions, classes, and parameters.

writePop(*outFile*, *exploreIter*, *pop*)
Writes a tab delimited text file outputting the entire evolved rule population, including conditions, phenotypes, and all rule parameters.

Parameters

- **outFile** –
- **exploreIter** –
- **pop** –

Returns

writePopStats(*outFile*, *trainEval*, *testEval*, *exploreIter*, *pop*, *correct*)
Makes output text file which includes all of the evaluation statistics for a complete analysis of all training and testing data on the current eLCS rule population.

Parameters

- **outFile** –
- **trainEval** –
- **testEval** –
- **exploreIter** –
- **pop** –
- **correct** –

Returns

1.10 eLCS.Prediction module

class eLCS.Prediction.**Prediction**(*population*)
Bases: object

Given a match set, this module uses a voting scheme to select the phenotype prediction.

Set up to handle both discrete and continuous phenotypes. Also set up to try and handle prediction ties if possible.

getDecision()
Returns prediction decision.

Returns

getFitnessSum(*population*, *low*, *high*)
Get the fitness sum of rules in the rule-set. For continuous phenotype prediction.

Parameters

- **population** –
- **low** –
- **high** –

Returns

1.11 eLCS.Timer module

class eLCS.Timer.**Timer**
Bases: object

Tracks and stores the run time of algorithm and some of it's major components

reportTimes()
Reports the time summaries for this run.

Returns a string ready to be printed out.

Returns The time summaries for the run

Return type str

returnGlobalTimer()
Set the global end timer, call at very end of algorithm

Returns The global time returned in minutes

Return type float

setTimerRestart(*remakeFile*)
Sets all time values to the those previously evolved in the loaded popFile

Parameters **remakeFile** (*str*) – File path to the remakeFile

startTimeDeletion()
Tracks Deletion Time

startTimeEvaluation()
Tracks Evaluation Time

startTimeMatching()
Tracks MatchSet Time

startTimeSelection()
Tracks Selection Time

startTimeSubsumption()
Tracks Subsumption Time

stopTimeDeletion()
Tracks Deletion Time

stopTimeEvaluation()
Tracks Evaluation Time

stopTimeMatching()
Tracks MatchSet Time

stopTimeSelection()
Tracks Selection Time

stopTimeSubsumption()
Tracks Subsumption Time

1.12 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

T

U

W