Cartesian

Release 0.1.10

Sep 17, 2018

Contents

cartesian: is a lightweight implementation of Cartesian genetic programming with symbolic regression in mind.

It is meant to be used in conjunction with deap or glyph.

The basic components are provided:

- data structure
- 1+4 Algorithm
- symbolic, ephemeral random and structure-based constants

CHAPTER 1

Installation

cartesian is available on PyPI

```
pip install cartesian
```

CHAPTER 2

API Documentation

2.1 cartesian

2.1.1 cartesian package

Submodules

cartesian.algorithm module

cartesian.algorithm.**return_opt_result**(*f*, *individual*)
Ensure that f returns a scipy.optimize.OptimizeResults

Parameters

- **f** – callable(individual)
- **individual** – instance of cartesian.cgp.Base

Returns OptimizeResult

cartesian.algorithm.**oneplus**(*fun*, *random_state=None*, *cls=None*, *lambda_=4*, *n_mutations=1*, *mutation_method='active'*, *maxiter=100*, *maxfev=None*, *f_tol=0*, *n_jobs=1*, *seed=None*, *callback=None*)
1 + lambda algorithm.

In each generation, create lambda offspring and compare their fitness to the parent individual. The fittest individual carries over to the next generation. In case of a draw, the offspring is prefered.

Parameters

- **fun** – callable(individual), function to be optimized
- **random_state** – an instance of np.random.RandomState, a seed integer or None
- **cls** – base class for individuals
- **lambda** – number of offspring per generation

- **n_mutations** – number of mutations per offspring
- **mutation_method** – specific mutation method
- **maxiter** – maximum number of generations
- **maxfev** – maximum number of function evaluations. Important, if fun is another optimizer
- **f_tol** – absolute error in metric(ind) between iterations that is acceptable for convergence
- **n_jobs** – number of jobs for joblib embarrassingly easy parallel
- **seed** – (optional) can be passed instead of cls, used for hot-starts
- **callback** – callable(OptimizeResult), can be optionally used to monitor progress

Returns scipy.optimize.OptimizeResult with non-standard attributes res.x = values for constants res.expr = expression res.fun = best value for the function

cartesian.algorithm.**optimize**(*fun*, *individual*)
Prepares individual and fun to optimize fun(c | individual)

Parameters

- **fun** – callable of lambda expression and its constant values.
- **individual** –

Returns scipy.optimize.OptimizeResult

cartesian.algorithm.**optimize_constants**(*fun*)
Wrap a measure with constant optimization.

cartesian.cgp module

class cartesian.cgp.**Primitive**(*name*, *function*, *arity*)
Bases: object

Basic build block for cartesian programs.

Parameters

- **name** – for text representation
- **function** –
- **arity** –

class cartesian.cgp.**Symbol**(*name*)
Bases: *cartesian.cgp.Primitive*

Base class for variables.

Will always be used in the boilerplate ensuring a uniform signature. Even if variable is not used in the genotype.

Parameters **name** – name of the primitive

class cartesian.cgp.**Constant**(*name*)
Bases: *cartesian.cgp.Symbol*

Base class for variables.

Will always be used in the boilerplate ensuring a uniform signature. Even if variable is not used in the genotype.

Parameters **name** – name of the primitive

class cartesian.cgp.**Ephemeral**(*name*, *fun*)
Bases: *cartesian.cgp.Primitive*

Base class for ERC's.

ERC's are terminals, but they are implemented as zero arity functions, as they do not need to appear in the argument list of the lambda expression.

Note: Compilation behaviour: Each individual has a dict to store its numeric values. Each position in the code block will only execute the function once. Values are lost during copying.

Parameters

- **name** – for text representation
- **fun** – callback, should return a random numeric values.

class cartesian.cgp.**Structural**(*name*, *function*, *arity*)
Bases: *cartesian.cgp.Primitive*

Structural constants are operators which take the graph representation of its arguments and convert it to a numeric value.

Parameters

- **name** – for text representation
- **function** –
- **arity** –

class cartesian.cgp.**PrimitiveSet**(*operators: list*, *terminals: list*, *mapping: dict*, *imapping: dict*, *context: dict*, *symbols: list*, *max_arity: int*)
Bases: object

A container holding the primitives and pre-compiled helper attributes.

Parameters

- **operators** – all non-terminal primitives (arity > 0)
- **terminals** – all terminals
- **max_arity** – maximum arity of all terminals. Determines the number of links for each register
- **mapping** – sorted and indexed list of the primitive set
- **imapping** – inverse of mapping
- **context** – links names of primitives to their functions
- **symbols** – all sybolic constants

classmethod create(*primitives*)
Create immutable PrimitiveSet with some attributes for quick lookups

class cartesian.cgp.**Base**(*code*, *outputs*)
Bases: sklearn.base.TransformerMixin

mapping
Helper dictionary to index the cartesian registers.

len_subgraph(*idx*)
Compute the length of the subgraph with head-node a idx.

active_genes
Computes the set of active gene in an individual.

clone()
Save copy, discard memory to refresh random constants

static format(*x*)

fit(*x*, *y=None*, ***fit_params*)

transform(*x*, *y=None*)

classmethod create(*random_state=None*)
Creates a new individual.

Each gene is picked with a uniform distribution from all allowed inputs or functions.

Parameters **random_state** – an instance of np.random.RandomState, a seed integer or None

Returns a new (random) individual

class cartesian.cgp.**Cartesian**(*name*, *primitive_set*, *n_columns=3*, *n_rows=1*, *n_back=1*, *n_out=1*)
Bases: type

Meta class to set class parameters and primitive set.

cartesian.cgp.**mutate**(*individual*, *n_mutations=1*, *method='active'*, *random_state=None*)
Create offsprings by mutating an individual.

Parameters

- **individual** – instance of Base
- **n_mutations** – number of mutations
- **method** – specific mutation method
- **random_state** – an instance of np.random.RandomState, a seed integer or None

Returns mutated individual

cartesian.cgp.**active_gene_mutation**(*individual*, *random_state=None*)
Picks an active gene and

Parameters

- **individual** – instance of Base
- **random_state** – an instance of np.random.RandomState, a seed integer or None

Returns mutated individual

cartesian.cgp.**point_mutation**(*individual*, *random_state=None*)
Picks a gene at random and mutates it.

The mutation is either rewiring, i.e. changing the inputs, or changing the operator (head of gene).

Parameters

- **individual** – instance of Base
- **random_state** – an instance of np.random.RandomState, a seed integer or None

Returns mutated individual

cartesian.cgp.**to_polish**(*c*, *return_args=True*)
Generates the polish notation of expression encoded by c.

Resolves the outputs recursively.

Note: Function has side-effects on the individual c. See Symbols for details.

Parameters

- **c** – instance of base
- **return_args** – optionally return the used arguments too

Returns polish notation of expression encoded by c

cartesian.cgp.**compile**(*c*)
Transform an individual into a lambda function

Parameters **c** – instance of Base

Returns lambda function

cartesian.sklearn_api module

class cartesian.sklearn_api.**Symbolic**(*operators=None*, *n_const=0*, *n_rows=1*, *n_columns=3*, *n_back=1*, *n_mutations=3*, *mutation_method='active'*, *maxiter=1000*, *maxfev=10000*, *lambda_=4*, *f_tol=0*, *seeded_individual=None*, *random_state=None*, *n_jobs=1*, *metric=None*, *callback=None*)
Bases: sklearn.base.BaseEstimator, sklearn.base.RegressorMixin

Wraps the 1 + lambda algorithm in sklearn api.

Note: n_costs provides a convenience method to create Symbols. All constants can be directly passed via the operators.

Parameters

- **operators** – list of primitives
- **n_const** – number of symbolic constants
- **n_rows** – number of rows in the code block
- **n_columns** – number of columns in the code block
- **n_back** – number of rows to look back for connections
- **n_mutations** – number of mutations per offspring
- **mutation_method** – specific mutation method
- **maxiter** – maximum number of generations
- **maxfev** – maximum number of function evaluations. Important, if fun is another optimizer
- **lambda** – number of offspring per generation
- **f_tol** – Absolute error in metric(ind) between iterations that is acceptable for convergence

- **seeded_individual** – an individual used to hot-start the optimization
- **random_state** – an instance of np.random.RandomState, an integer used as seed, or None
- **n_jobs** – number of jobs for joblib embarrassingly easy parallel
- **metric** – callable(individual), function to be optimized
- **callback** – callable(OptimizeResult), can be optionally used to monitor progress

fit (*x*, *y*)

Trains the model given the regression task.

Parameters

- **x** (*np.ndarray*) – input data matrix for fitting of size (number_of_input_points, number_of_features)
- **y** (*np.ndarray*) – target data vector for fitting of size (number_of_input_points)

Returns self

predict (*x*)

Use the fitted model f to make a prediction.

Parameters **x** – input data matrix for scoring

Returns predicted target data vector

cartesian.util module

cartesian.util.**make_it** (*x*)

Ensures that x is an iterator.

If x is not iterable, wrap it as a one-elemened tuple.

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index