pushgp.py Documentation

Release 0.1.0

Saul Shanabrook

May 07, 2014

Contents

Use genetic programming, implemented with Push, for machine learning.

pip install -e .

- Free software: BSD license
- Documentation: http://pushgp.rtfd.org.

Contents:

CHAPTER 1

Installation

Only works on Python 3.4+

At the command line:

```
$ git clone https://github.com/saulshanabrook/pushgp.py.git
$ cd pushgp.py
$ pip install -e .
```

CHAPTER 2

pushgp

PushGP is really a collection of various interlocking piece, enabling you to run symbolic regression using the Push programming language.

At the most high level is `pushgp.estimators.PushGPRegression`, built on scikit-learn. This allows you to use genetic programming in your machine learning toolchain, as you would any other technique, such as linear regression.

Then there is `pushgp.ec.ec.PushGP` which performs the actual evolution of Push programs. This is implemented as an evolutionary computation engine for Inspyred.

That module, in turn, makes use of `pushgp.push.interpreter.Push`, which handles all of the language logic.

Right now, the only part to be fully implemented and test is the `Push` interpreter.

2.1 pushgp.estimators

Warning: This module is not fully implemented yet. It will not work as is.

2.2 pushgp.ec

2.2.1 pushgp.ec.ec

Warning: This module is not fully implemented yet. It will not work as is.

class pushgp.ec.ec.**PushGP**(*random*)

Bases: inspyred.ec.ec.EvolutionaryComputation

Evoluiontary computation representing genetic programming using Push

Optional keyword arguments in `evolve`:

- *pop_size* – the number of individuals to be created
- *mutation_rate* – the rate at which mutation is performed (default 0.1)
- *crossover_rate* – the rate at which crossover is performed (default 1.0)
- *push_max_size* – the maximum number of instructions in a Push program (default 500)

•*push_instructions* – a list of possible push intructions to use

(default [])

__init__(*random*)

__module__ = 'pushgp.ec.ec'

evolve(**args*, ***kwargs*)

2.3 pushgp.push

```
>>> from pushgp.push.interpreter import Push
>>> from pushgp.push.instructions import bool, float, str
>>> p = Push()
>>> p(1.0, 3.0)
Push: {'exec': [], 'float': [3.0, 1.0]}
>>> p(float.div)
Push: {'exec': [], 'float': [0.3333333333333333]}
>>> p(True, False)
Push: {'exec': [], 'float': [0.3333333333333333], 'bool': [False, True]}
>>> p(bool.or_)
Push: {'exec': [], 'float': [0.3333333333333333], 'bool': [True]}
>>> p(bool.from_float)
Push: {'exec': [], 'float': [], 'bool': [True, True]}
```

2.3.1 pushgp.push.interpreter

class pushgp.push.interpreter.**Push**

Bases: collections.defaultdict

The Push interpreter.

The different stack types are available as dictionary keys and values. For example:

```
>>> p = Push()
>>> p['exec'] = [1, 2]
>>> p
Push: {'exec': [1, 2]}
```

Each stack is implemented as a List, so any method that work on lists will work on the stacks. For example, to pop an item from a stack use the `.pop()` method.

```
>>> p = Push()
>>> p['exec'] = [1, 2]
>>> p['exec'].pop()
2
>>> p
Push: {'exec': [1]}
```

As you can see, the top of stack is the last item in the list.

To execute the `exec` stack, call `execute()` on an instance.

```
>>> p = Push()
>>> p['exec'] = [1, 2]
>>> p.execute()
```

```
>>> p
Push: {'int': [2, 1], 'exec': []}
```

To push an item to the `exec` stack and then execute it, just call the initiated object on any number of items.

```
>>> Push()(1, 2)
Push: {'exec': [], 'int': [2, 1]}
```

You can even chane these calls together.

```
>>> Push()(1)(2)
Push: {'exec': [], 'int': [1, 2]}
```

__call__(**items*)
Pushes a list of `items` onto the `exec` stack and executes them. Uses the ._execute() method to run through the `exec` stack.

__dict__ = dict_proxy({'__module__': 'pushgp.push.interpreter', '__str__': <function __str__ at 0x28bf230>, 'execute'

__init__()

__module__ = 'pushgp.push.interpreter'

__repr__()

__str__()

execute()
Pops each item off the `exec` stack until it is empty, using the *.pop()* method.

If the item is in instance in the `stack_types` attribute then it will be pushed to the corresponding stack.

Else if it a callable is will be called with the Push object and should return a modified Push object.

Else it will raise a `TypeError`

stack_for_item(*item*)
Returns the right stack name for any `item`, using the `stack_types` map attribute. If no stack has this type, it will return None.

2.3.2 pushgp.push.instructions

pushgp.push.instructions.bool

pushgp.push.instructions.float

pushgp.push.instructions.str

p