archery Documentation

Release 1.1.1

Julien HawkeEye Tayon

Oct 30, 2018

Contents

- Source : https://github.com/jul/archery
- Tickets : https://github.com/jul/archery/issues?state=open
- Latest documentation : http://archery.readthedocs.org/en/latest/index.html

It is set of Mixins to use on MutableMapping giving the following features :

- Linear Algebrae;
- Vector like metrics;
- Searchable behaviour;

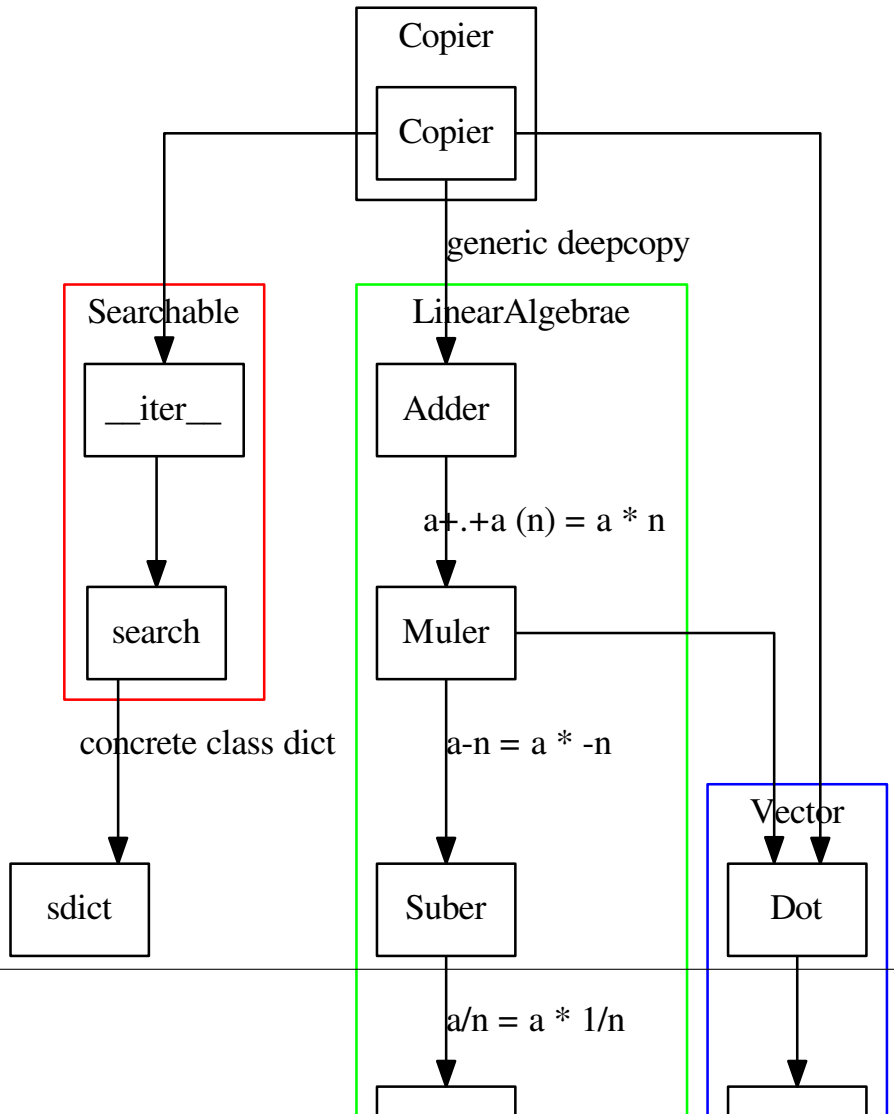
for convenience 3 concrete classes are provided :

- *mdict* (dict that follow the rules of linear algebrae based on dict);
- *vdict* (dict that have cos, abs, dot product);
- *sdict* (dict that are easily searchable);

following this inheritance graph of traits

CHAPTER 1

Graph



CHAPTER 2

Basic Usage

Using the ready to use class derived from dict

2.1 mdict

dict that supports consistently all the linear algebrae properties

Basically : dict that are vectors on arbitrary basis (recursively).

To learn more about its use and implementation:

- Video presentation in FOSDEM 2017
- or look at the presentation

ex:

```
>>> from archery import mdict
>>> point = mdict(x=1, y=1, z=1)
>>> point2 = mdict(x=1, y=-1)
>>> print( (2 * point + point2)/4)
>>> # OUT : {'y': 0.25, 'x': 0.75, 'z': 0.5}
>>> print(point - point2)
>>> # OUT : {'y': 2, 'x': 0, 'z': 1}
>>> b=mdict(x=2, z=-1)
>>> a=mdict(x=1, y=2.0)
>>> a+b
>>> # OUT: {'y': 2.0, 'x': 3, 'z': -1}
>>> b-a
>>> # OUT: {'y': -2.0, 'x': 1, 'z': -1}
>>> -(a-b)
>>> # OUT: {'y': -2.0, 'x': 1, 'z': -1}
>>> a+1
>>> # OUT: {'y': 3.0, 'x': 2}
>>> -1-a
```

(continues on next page)

(continued from previous page)

```
>>> # OUT: {'y': -3.0, 'x': -2}
>>> a*b
>>> # OUT: {'x': 2}
>>> a/b
>>> # OUT: {'x': 0}
>>> 1.0*a/b
>>> # OUT: {'x': 0.5}
```

2.2 vdict

dict that defines *abs()*, *dot()*, *cos()* in the euclidean meaning

ex::

```
>>> from archery import vdict as Point
>>>
>>> u = Point(x=1, y=1)
>>> v = Point(x=1, y=0)
>>> u.cos(v)
>>> 0.7071067811865475
>>> u.dot(v)
>>> # OUT: 1
>>> u.cos(2*v)
>>> # OUT: 0.7071067811865475
>>> u.dot(2*v)
>>> #OUT: 2
>>> abs(u)
>>> #OUT: 1.4142135623730951
>>> u3 = Point(x=1, y=1, z=2)
>>> u4 = Point(x=1, y=3, z=4)
>>> u3 + u4
>>> #OUT: dict(x=2, y=4, z=6)
>>> assert u4 + u4 == 2*u4
>>> from archery import vdict
>>> from math import acos, pi
>>> point = vdict(x=1, y=1, z=1)
>>> point2 = vdict(x=1, y=-1)
>>> point2 = mdict(x=1, y=-1)
>>> print( (2 * point + point2)/4)
>>> # OUT : {'y': 0.25, 'x': 0.75, 'z': 0.5}
>>> print(acos(vdict(x=1,y=0).cos(vdict(x=1, y=1)))*360/2/pi)
>>> # OUT : 45.0
>>> print(abs(vdict(x=1, y=1)))
>>> # OUT : 1.41421356237
>>> print(vdict(x=1,y=0,z=3).dot(vdict(x=1, y=1, z=-1)))
>>> #OUT -2
```

2.3 sdict

dict made for searching value/keys/*Path* with special interests.

Basically, it returns an interator in the form of a tuple being all the keys and the value. It is a neat trick, if you combine it with *make_from_path*, it helps select exactly what you want in a dict:

```
>>> from archery import sdict, make_from_path
>>> tree = sdict(
...       a = 1,
...       b = dict(
...           c = 3.0,
...           d = dict(e=True)
...       ),
...       point = dict( x=1, y=1, z=0),
... )
>>> list(tree.leaf_search(lambda x: type(x) is float ))
>>> #Out: [3.0]
>>> res = list(tree.search(lambda x: ("point") in x ))
>>> ## equivalent to list(tree.search(lambda x: Path(x).contains("point")))
>>> print(res)
>>> #Out: [('point', 'y', 1), ('point', 'x', 1), ('point', 'z', 0)]
>>> sum([ make_from_path(mdict, r) for r in  res])
>>> #Out:  {'point': {'x': 1, 'y': 1, 'z': 0}}
```

CHAPTER 3

Advanced usage

This library is a proof of the consistent use of Mixins on MutableMapping gives the property seen in the basic usage.

The Mixins do not require any specifics regarding the implementation and **should** work if I did my job properly with any kinds of *MutableMapping*.

Here is an example of a cosine similarities out of the box on the *Collections.Counter*

```
>>> from collections import Counter
>>> from archery import VectorDict
>>> class CWCos(VectorDict, Counter):
...     pass
>>>
>>> CWCos(["mot", "wut", "wut", "bla"]).cos(CWCos(["mot","wut", "bla"]))
>>> # OUT: 0.942809041582
```

You can also inherit LinearAlgebrae

CHAPTER 4

API

4.1 VectorDict / vdict

class archery.trait.**Vector**

__abs__()
return the absolute value (hence >=0) aka the distance from origin as defined in Euclidean geometry. Keys of the dict are the dimension, values are the metrics https://en.wikipedia.org/wiki/Euclidean_distance

cos(*v*)
returns the cosine similarity of 2 mutable mappings (recursive) https://en.wikipedia.org/wiki/Cosine_similarity dict().cos(dict(x=....)) will logically yield division by 0 exception. http://math.stackexchange.com/a/932454

dot(*v*)
scalar product of two MappableMappings (recursive) https://en.wikipedia.org/wiki/Dot_product

4.2 Searchable, sdict

class archery.trait.**Searchable**

leaf_search(*predicate*)
Return a generator all all values matching the predicates

search(*predicate*)
Return a generator of all tuples made of : - all keys leading to a value - and the value itself that match the predicate on the *Path*

4.3 Path

Basically a class meant for making search in *sdict* more readable so that you have shortcuts that are more meaningfull than manipulating a tuple

class archery.**Path**

contains(**a_tuple*)
checks if the serie of keys is contained in a path

```
>>> p = Path( [ 'a', 'b', 'c', 'd' ] )
>>> p.contains( 'b', 'c' )
>>> True
```

endswith(**a_tuple*)
check if path ends with the consecutive given has argumenbts value

```
>>> p = Path( [ 'a', 'b', 'c' ] )
>>> p.endswith( 'b', 'c' )
>>> True
>>> p.endswith( 'c', 'b' )
>>> False
```

key()
function provided for code readability: - returns all the keys in the Path

startswith(**a_tuple*)
checks if a path starts with the value

```
>>> p = Path( [ 'a', 'b', 'c', 'd' ] )
>>> p.startswith( 'a', 'b' )
>>> True
```

value()
function provided for code readability: - returns the left most value of the Path aka the value

4.4 make_from_path

Making dict great vectors!

archery.**make_from_path**(*type_of_mapping*, *path*)
Work in Progress create a mutable mapping from a *Path* (tuple made of a series of keys in a dict leading to a value followed by a value). The source is used a mapping factory and is reset in the process

```
>>> make_from_path(dict, ("y", "z", 2))
>>> #Out[2]: {'y': {'z': 2}}
```

4.5 mapping_row_iter

Making dict great vectors!

archery.**mapping_row_iter**(*tree*, *path=<object object>*)

iterator on a tree that yield an iterator on a mapping in the form of a list of ordered key that leads to the element and the value

```
>>> from archery import mapping_row_iter
>>> [ x for x in mapping_row_iter({
...         "john" : {'math':10.0, 'sport':1.0},~
...         "lily" : { 'math':20, 'sport':15.0}
...     })]
>>> #[['john', 'sport', 1.0], ['john', 'math', 10.0],~
>>> #['lily', 'sport', 15.0], ['lily', 'math', 20]]
```

CHAPTER 5

Detailed documentation

Contents:

5.1 Having fun

5.1.1 Mixing scalars and records (side effect)

You can also the use the addition in the meaning of a record. That is what the yahi module on pypi does https://github.com/jul/yahi

```
>>> 2*mdict(x=1, y="lo",z=[2])
{'y': 'lolo', 'x': 2, 'z': [2, 2]}
>>> mdict(y=1, z=1)*Daikyu(x=1, y="lo",z=[2])*2
{'y': 'lolo', 'z': [2, 2]}
>>> a=mdict(dictception=dict(a=1,b=2), sample = 1, data=[1,2])
>>> b=mdict(dictception=dict(c=-1,b=2), sample = 2, data=[-1,-2])
>>> a+b
{'sample': 3, 'dictception': {'a': 1, 'c': -1, 'b': 4}, 'data': [1, 2, -1, -2]}
>>> mdict(dictception=1, sample=1)* a*b
{'sample': 2, 'dictception': {'b': 4}}
```

5.1.2 Pushing the vice to create a rotation matrix with a dict

```
#!/usr/bin/env python3
from archery import mdict, vdict
from math import pi, cos, sin, acos

class Matrix(mdict):
    def __call__(self, other):
        other = other.copy()
```

(continues on next page)

(continued from previous page)

```
        res= vdict()
        for (src, dst), functor in self.items():
            res += mdict({ dst: functor(other[src])})
        return res


theta = pi/6

u = mdict(x=1, y=2)
v = mdict(x=1, y=0)
alien = vdict(x=u, y=v)

def rotation_maker(theta):
    """Matrix takes as key (SRC, DST) (which is the opposite of "actual notation")
    """
    return Matrix({
        ("x", "x") : lambda v:1.0 *  v * cos(theta),
        ("y", "x") : lambda v:1.0 * -v * sin(theta),
        ("x", "y") : lambda v:1.0 *  v * sin(theta),
        ("y", "y") : lambda v:1.0 *  v * cos(theta)
    })

rotation = rotation_maker(pi/6)

print(u)
# OUT:{'x': 1, 'y': 2}
print(rotation(u))
# OUT:{'x': -0.13397459621556118, 'y': 2.232050807568877}
print("*" * 80)
# OUT:********************************************************************************
print(v)
# OUT:{'x': 1, 'y': 0}
print(rotation(v))
# OUT:{'x': 0.8660254037844387, 'y': 0.4999999999999994}
print(acos(vdict(v).cos(vdict(rotation(v))))/2 / pi * 360)
# OUT:29.999999999999993
print(acos(vdict(v).cos(vdict(rotation_maker(pi/3)(v))))/2 / pi * 360)
# OUT:60.0
print(acos(vdict(v).cos(vdict(rotation_maker(pi/5)(v))))/2 / pi * 360)
# OUT:36.0
print(alien)
print(acos(alien.cos(rotation_maker(pi/4)(alien)))/2 / pi * 360)
print(alien)
print(rotation_maker(pi/4)(alien))
print(alien)
print(u)
print(v)
```

5.2 Design

Traits are Mixins, behaviours. All these terms recovers loosely the same idea.

In this case refering to even older conventions traits are concrete classes for abstract classes/interfaces.

collections.MutableMapping defines an interface and some concrete methods. Since isinstance relies on interfaces

(ducktyping) I can safely use it to implement methods that don't exists and will normally work for most Mappings.

5.2.1 Quivers : consistent sets of Traits

Note: Yes, it is a pun, trait = arrow <=> quiver = set of arrows.

5.2.2 Inclusive Trait

If a key is absent on one of the Mapping, it will be considered the neutral element. An empty list, for list, 0 for int, 0.0 for float...

The behaviour of addition and substraction is consistently deriving from the boolean algebrae meaning of **+** in a set context where + means union.

Thus Addition and substraction are inclusive.

5.2.3 Exclusive Trait

Multiplication operates as an intersection, because on one hand it is consistentwith the set/boolean meaning of multiplication, and also that neutral element of addition, is normaly the null element of multiplication. Since multiplication implies division, instead of multiplying by 0 and keeping present in at least one dict, I prefer to avoid the raging division by zero. In short, I try to avoid my dict to explode when dividing by 0. I am weak I know.

5.2.4 Summary of the behaviours and dependancies

Operation	Short	Behaviour	Requires	Safe	Name
Copier	copy	None			
Addition	add	Inclusive	copy	Yes	InclusiveAdder
Multiplication	mul	Exclusive	add,copy	Yes	ExclusiveMuler
Substraction	sub	Inclusive	add,mul,copy	Yes	InclusiveSubber
Division	div	Exclusive	add,mul,sub,copy	No	TaintedExclusiveDiver

5.3 What is addition in MutableMapping useful for?

It is used with yahi as an exemple. I find addition on MutableMapping a very convenient way to reduce by using in place addition (__iadd__).

VectorDict also has an exemple of map/reduce with multiprocessing word counting

MapReduce is a way of treating big data without consuming too much memory ensuring relativley good performance. It is normaly considered to belong to the functional paradigm and is best used with generators.

5.4 Changelog and roadmap

5.4.1 Changelog

1.1.1 Trying very hard to have the README.rst formated.

1.1.0 *make_from_path* : it made no sense it took a first argument a MutableMapping that would be destroyed in the process. Now takes a type of MutableMapping as an input.

1.0.0 Flatter and simpler naming (while keeping descendant compatibility)

0.1.8 release with better code coverage

0.1.7 Maintenance release correcting minor bugs in preparation for the 1.0 release

0.1.6 Tested py3.2 on my freeBSD, it works for me ©

0.1.4 closes #6 : trying to install on debian stable is like contemplating a machine frozen 5 years ago. Rerunning tests on debian

0.1.3 blocking install if tests don't pass

0.1.2 py3 compliance

0.1.1 closing issue in iadd: some performance issue in __iadd__ aka +=

0.1.0 initial release

5.4.2 Convention:

version x.y.z

while in beta convention is :

- **x** = 0
- **y** = API change
- **z** = bugfix and/or improvement

and then

- **x** = API change
- **y** = improvement
- **z** = bugfix

5.4.3 Roadmap

1.0.0

- Flattening the structure of archery and making naming more obvious
- Keeping the old API compatible
- Begining deprecation
- maybe prepare a set of trait to make recursive dict looks like *sets* in a consistent way

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

Symbols

A

C

D

E

K

L

M

P

S

V