pybloomfiltermmap3 documentation

Release 0.5.2

Michael Axiak, Prashant Sinha, Vytautas Mizgiris and others

Jan 13, 2020

CLASS REFERENCE

1 Why pybloomfilter? **3**

2 Install **5**

3 Develop **7**

3.1 BloomFilter Class Reference 7

3.2 License 11

3.3 Authors 12

3.4 Changelog 12

Python Module Index **15**

Index **17**

pybloomfiltermmap3 is a Python 3 fork of *pybloomfiltermmap* by Michael Axiak (@axiak).

Bloom filter is a probabilistic data structure used to test whether an element is a member of a set. The wikipedia page has further information on their nature.

This module implements a Bloom filter in Python that's fast and uses mmap files for better scalability.

Here's a quick example:

```
>>> from pybloomfilter import BloomFilter

>>> bf = BloomFilter(10000000, 0.01, 'filter.bloom')
>>> with open("/usr/share/dict/words") as f:
>>>     for word in f:
>>>         bf.add(word.rstrip())

>>> print 'apple' in bf
True
```

That wasn't so hard, was it? Now, there are a lot of other things we can do. For instance, let's say we want to create a similar filter with just a few pieces of fruit:

```
>>> fruitbf = bf.copy_template("fruit.bloom")
>>> fruitbf.update(("apple", "banana", "orange", "pear"))

>>> print(fruitbf.to_base64())
"eJzt2k13ojAUBuA9f8WFyofF5TWChlTHaPzqrlqFCtj6gQi/frqZM2N7aq3Gis59d2ye85KTRbhk"
"0lyu1NRmsQrgRda0I+wZCfXIaxuWv+jqDxA8vdaf21HIOSn1u6LRE0VL9Z/qghfbBmxZoHsqM3k8"
"N5XyPAxH2p22TJJoqwU9Q0y0dNDYrOHBIa3BwuznapG+KZZq69JUG0zu1tqI5weJKdpGq7PNJ6tB"
"GKmzcGWWy8o0FeNNYNZAQpSdJwajt7eRhJ2YM2NOkTnSsBOCGGKIIYbY2TA663GgWWyWfUwn3oIc"
"fyLYxeQwiF07RqBg9NgHrG5ba3jba5yl4zS2LtEMMcQQQwwxmRiBhPGOJOywIPafYhUwqnTvZOfY"
"Zu40HH/YxDexZojJwsx6ObDcT7D8vVOtJBxiAhD/AjMmjeF2Wnqd+5RrHdo4azPEzoANabiUhh0b"
"xBBDDDHEENsf8twlrizswEjDhnTbzWazbGKpQ5k07E9Ox2iFvXBZ2D9B7DawyqLFu5lshhhiiGUK"
"a4nUloa9yxkwR7XhgPPXYdhRIa77uDtnyvqaIXalGK02ufv3J36GmsnG4lquPnN9gJo1VNxqgYbt"
"ji/EC8s1PWG5fuVizW4Jox6/3o9XxBBDDLFbwcg9v/AwjrPHtTRsX34O01mxLw37bhCTjJk0+PLK"
"08HYd4MYYojdKmYnBfjsktEpySY2tGGZzWaIIfYDGB271Yaieaat/AaOkNKb"
```

CHAPTER

ONE

WHY PYBLOOMFILTER?

As already mentioned, there are a couple reasons to use this module:

- It natively uses mmaped files.
- It natively does the set things you want a Bloom filter to do.
- It is fast (see benchmarks).

CHAPTER

TWO

INSTALL

Please note that this version is for Python 3.5 and over. In case you are using Python 2, please see pybloomfiltermmap.

To build and install:

```
$ pip install pybloomfiltermmap3
```

CHAPTER

THREE

DEVELOP

To develop you will need Cython. The setup.py script should automatically build from Cython source if the Cython module is available.

3.1 BloomFilter Class Reference

class pybloomfilter.**BloomFilter**(*capacity: int, error_rate: float*[, *filename = None: string*][, *perm=0755*][, *hash_seeds = None: list*])

Creates a new BloomFilter object with a given capacity and error_rate.

Parameters

- **capacity** (*int*) – the maximum number of elements this filter can contain while keeping the false positive rate under `error_rate`.
- **error_rate** (*float*) – false positive probability that will hold given that `capacity` is not exceeded.
- **filename** (*str*) – filename to use to create the new Bloom filter. If a filename is not provided, an in-memory Bloom filter will be created.
- **perm** (*int*) – (*not applicable for an in-memory Bloom filter*) file access permission flags.
- **hash_seeds** (*list*) – optionally specify hash seeds to use for the hashing algorithm. Each hash seed must not exceed 32 bits. The number of hash seeds will determine the number of hashes performed.

Note that we do not check capacity. This is important, because we want to be able to support logical OR and AND (see `BloomFilter.union()` and `BloomFilter.intersection()`). The capacity and error_rate then together serve as a contract – you add less than capacity items, and the Bloom filter will have an error rate less than error_rate.

Raises `OSError` if the supplied filename does not exist or if user lacks permission to access such file. Raises `ValueError` if the supplied `error_rate` is invalid, `hash_seeds` does not contain valid hash seeds, or if the file cannot be read.

3.1.1 Class Methods

classmethod BloomFilter.**open**(*filename*[, *mode="rw"*])

Creates a `BloomFilter` object from an existing file.

Parameters

- **filename** (*str*) – existing filename

- **mode** (*str*) – file access mode

Return type *BloomFilter*

classmethod BloomFilter.**from_base64**(*filename*, *string*[, *perm=0755*])

Unpacks the supplied base64 string (as returned by *BloomFilter.to_base64()*) into the supplied filename and return a *BloomFilter* object using that file.

Example:

```
>>> bf = BloomFilter.from_base64("/tmp/mike.bf",
    "eJwFwcuWgiAAANC9v+JCx7By0QKt0GHEbKSknflAQ9QmTyRfP/fW5E9XTRSX"
    "qcLlqGNXphAqcfVH\nRoNv0n4JlTpIvAP0e1+RyXX6I637ggA+VPZnTYR1A4"
    "Um5s9geYaZZLiT208JIiG3iwhf3Fwlzb3Y\n5NRL4uNQS6/d9OvTDJbnZMnR"
    "zcrplOX5kmsVIkQziM+vw4hCDQ3OkN9m3WVfPWzGfaTeRftMCLws\nPnzEzs"
    "gjAW60xZTBbj/bOAgYbK50PqjdzvgHZ6FHZw==\n")
>>> "MIKE" in bf
True
```

Parameters

- **filename** (*str*) – new filename
- **perm** (*int*) – file access permission flags

Return type *BloomFilter*

3.1.2 Instance Attributes

BloomFilter.capacity -> int

The maximum number of elements this filter can contain while keeping the false positive rate under `BloomFilter.error_rate`. Returns an integer.

BloomFilter.error_rate -> float

The acceptable probability of false positives. Returns a float.

BloomFilter.bit_array -> int

Bit vector representation of the Bloom filter contents. Returns an integer.

BloomFilter.hash_seeds -> list

Integer seeds used for the random hashing. Returns a list of integers.

BloomFilter.filename -> string

File name (compatible with file objects). Does not apply to an in-memory *BloomFilter* and will raise `ValueError` if accessed. Returns a string.

BloomFilter.num_bits -> int

Number of bits used in the filter as buckets. Returns an integer.

BloomFilter.num_hashes -> int

Number of hash functions used when computing. Returns an integer.

BloomFilter.read_only -> bool

Indicates if the opened *BloomFilter* is read-only. Always `False` for an in-memory *BloomFilter*.

BloomFilter.name -> bytes

PENDING DEPRECATION: use `BloomFilter.filename()` instead.

File name (compatible with file objects). Does not apply to an in-memory *BloomFilter* and will raise `ValueError` if accessed. Returns an encoded string.

3.1.3 Instance Methods

`BloomFilter.add(item)`

Adds an item to the Bloom filter. Returns a boolean indicating whether this item was present in the Bloom filter prior to adding (see `BloomFilter.__contains__()`).

Parameters **item** – hashable object

Return type bool

`Bloomilter.clear_all()`

Removes all elements from the Bloom filter at once.

`BloomFilter.copy(filename)`

Copies the current `BloomFilter` object to another object with a new filename.

Parameters **filename** (*str*) – new filename

Return type `BloomFilter`

`BloomFilter.copy_template(filename[, perm=0755])`

Creates a new `BloomFilter` object with the exact same parameters. Once this is performed, the two filters are comparable, so you can perform set operations using logical operators.

Example:

```
>>> apple = BloomFilter(100, 0.1, '/tmp/apple')
>>> apple.add('granny_smith')
False
>>> pear = apple.copy_template('/tmp/pear')
>>> pear.add('conference')
False
>>> pear |= apple
```

Parameters

- **filename** (*str*) – new filename
- **perm** (*int*) – file access permission flags

Return type `BloomFilter`

`BloomFilter.sync()`

Forces a `sync()` call on the underlying mmap file object. Use this if you are about to copy the file and you want to be sure you got everything correctly.

`BloomFilter.to_base64()`

Serializes the `BloomFilter` instance. Returns a compressed, base64 encoded string. This string can later be unpacked into a `BloomFilter` using `BloomFilter.from_base64()`.

This may also be used to compare filter contents, given that the same `error_rate`, `capacity` and `hash_seeds` were used when constructing such filters. For example:

```
>>> b64_repr =
↪"eJwFwUsOgjAUAMADuZCgKBsXhQeIWKRaEuquFihGPoYqDzm9M1U6LmUdU8UwUcNshM2IRssAwWfgSxjHjO6ssssn6bLsY
↪dgYSuqzZ1cwISL1YrcH9V9PQ3cdN/JuRqn6nkRynUtd8rpmkldMt7Kb5EfF5d/IEl1GP/
↪8LUuEYHN0HR5ihXL/1u65WKKZQkFsDykPfhQCpEAGGqexd4MX+vgkJ0/
↪LCHIRNXpL0rk8SXH4A2pERcg="
>>> hash_seeds = [3837895095, 3446164276, 218928576, 318812276, 2715048734,␣
↪4231234832, 2646234356, 1058991177, 1248068903, 1134013883, 3269341494,␣
↪3044656612, 3079736504]
```

(continues on next page)

(continued from previous page)

```
>>> bf = BloomFilter.from_base64("/tmp/bf", b64_repr)

>>> bf_rec = BloomFilter(bf.capacity, bf.error_rate, "/tmp/bf_rec", hash_seeds=bf.
↪hash_seeds.tolist())
>>> bf_rec.add("5f35c4edcdb5b970ac8939a3c7abb3347ed9c4e3e251cbc799bdaeba008ce7aa")
>>> bf_rec.add("f416d946d98166066611fb1a5e262c5f241d9bfdd8c885e062433b6f6b73799a")

>>> assert bf_rec.to_base64() == bf.to_base64()
```

Return type base64 encoded string representing filter

`BloomFilter.update(iterable)`

Calls `BloomFilter.add()` on all items in the iterable.

`BloomFilter.union(filter)`

Performs a set OR with another comparable filter. You can (only) construct comparable filters with `BloomFilter.copy_template()` above. In the above example, Bloom filter `pear` will have both "granny_smith" and "conference".

The computation will occur *in place*. That is, calling:

```
>>> bf.union(bf2)
```

is a way of adding *all* the elements of `bf2` to `bf`.

NB: Calling this function will render future calls to len() invalid.

Parameters **other** (`BloomFilter`) – filter to perform the union with

Return type `BloomFilter`

`BloomFilter.intersection(other)`

The same as `BloomFilter.union()` above except it uses a set AND instead of a set OR.

NB: Calling this function will render future calls to len() invalid.

Parameters **other** (`BloomFilter`) – filter to perform the intersection with

Return type `BloomFilter`

`BloomFilter.close()`

Closes the currently opened `BloomFilter` file descriptor. Following accesses to this instance will raise a `ValueError`.

Caution: this will delete an in-memory filter irrecoverably!

3.1.4 Magic Methods

`BloomFilter.__len__(item)`

Returns the number of distinct elements that have been added to the `BloomFilter` object, subject to the error given in `BloomFilter.error_rate`.

Example:

```
>>> bf = BloomFilter(100, 0.1, '/tmp/fruit.bloom')
>>> bf.add('apple')
>>> bf.add('apple')
```

(continues on next page)

(continued from previous page)

```
>>> bf.add('orange')
>>> len(bf)
2
```

Raises `IndeterminateCountError` if a the Bloom filter was a result of a set operation. Example:

```
>>> bf2 = bf.copy_template('/tmp/new.bloom')
>>> bf2 |= bf
>>> len(bf2)
Traceback (most recent call last):
    ...
pybloomfilter.IndeterminateCountError: Length of BloomFilter object is␣
↪unavailable after intersection or union called.
```

Parameters **item** – hashable object

Return type int

`BloomFilter.__contains__(item)`

Checks to see if item is contained in the filter, with an acceptable false positive rate of `BloomFilter.error_rate` (see above).

Parameters **item** – hashable object

Return type bool

`BloomFilter.__ior__(filter)`

See `BloomFilter.union()`.

`BloomFilter.__iand__(filter)`

See `BloomFilter.intersection()`.

3.1.5 Exceptions

class `pybloomfilter.IndeterminateCountError(message)`

The exception that is raised if len() is called on a `BloomFilter` object after |=, &=, `BloomFilter.intersection()`, or `BloomFilter.union()` is used.

3.2 License

MIT License

3.3 Authors

3.3.1 Current maintainers

- Prashant Sinha (@prashnts)
- Vytautas Mizgiris (@mizvyt)

3.3.2 Original author

- Michael Axiak (@axiak)

3.3.3 Contributors

- Rob Stacey
- dlecocq: for superfast addition
- pbutler: fix memory leak
- Dan Crosta: convert MurmurHash3 to C from C++
- gaetano-guerriero: fixed base64 dumps
- gonzalezzfelipe: fixed buggy "copy_template" method

3.4 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

3.4.1 0.5.2 (2020-01-13)

Changes

- Python setup will now always try to use and build from Cython, if the module is available in the current environment. To force cythonize, use "–cython". If the module is not available and no "–cython" was used, the setup will look for a bundled Cython source.

3.4.2 0.5.1 (2019-12-31)

Changes

- Add `BloomFilter.bit_array()` property for bit vector representation
- Add `BloomFilter.filename()` property and issue a PendingDeprecationWarning when using `BloomFilter.name()`
- Do memset after initializing BloomFilter instance to set alignment bytes to 0 prior to populating the filter (see notes in #24)
- Remove `mode` parameter from `BloomFilter.from_base64()` method introduced in 0.5.0 as part of a refactoring (see notes in #23)
- Add explicit flag to build using Cython when building or installing a package; setup looks for a bundled Cython source by default (included in the PyPI distribution package)

3.4.3 0.5.0 (2019-11-25)

Changes

- Add support for read-only Bloom filter files
- Add customization of hash seeds for hashing algorithms
- Drop Python < 3.5 support

3.4.4 0.4.19 (2019-10-11)

Changes

- Ensure that filename is encoded in *`copy_template()`* (thanks @gonzalezzfelipe!)

3.4.5 0.4.18 (2019-10-08)

Fixes

- Fix missing Cython dependency in setup.py

3.4.6 0.4.17 (2019-08-25)

Fixes

- PyPi wants `long_description` and its type

3.4.7 0.4.16 (2019-08-25)

Fixes

- Fix read / write of base64 encoded filter files (thanks @gaetano-guerriero!)

3.4.8 0.4.15 (2019-04-09)

Changes

- Remove Python 2 support, add Python 3 support

3.4.9 Previous Versions

See Python 2 pybloomfiltermmap CHANGELOG.

PYTHON MODULE INDEX

p

`pybloomfilter` (*Unix, Windows*), 7

INDEX

Symbols

`__contains__()` (*pybloomfilter.BloomFilter method*), 11
`__iand__()` (*pybloomfilter.BloomFilter method*), 11
`__ior__()` (*pybloomfilter.BloomFilter method*), 11
`__len__()` (*pybloomfilter.BloomFilter method*), 10

A

`add()` (*pybloomfilter.BloomFilter method*), 9

B

`BloomFilter` (*class in pybloomfilter*), 7

C

`clear_all()` (*pybloomfilter.Bloomilter method*), 9
`close()` (*pybloomfilter.BloomFilter method*), 10
`copy()` (*pybloomfilter.BloomFilter method*), 9
`copy_template()` (*pybloomfilter.BloomFilter method*), 9

F

`from_base64()` (*pybloomfilter.BloomFilter class method*), 8

I

`IndeterminateCountError` (*class in pybloomfilter*), 11
`intersection()` (*pybloomfilter.BloomFilter method*), 10

O

`open()` (*pybloomfilter.BloomFilter class method*), 7

P

`pybloomfilter` (*module*), 7

S

`sync()` (*pybloomfilter.BloomFilter method*), 9

T

`to_base64()` (*pybloomfilter.BloomFilter method*), 9

U

`union()` (*pybloomfilter.BloomFilter method*), 10
`update()` (*pybloomfilter.BloomFilter method*), 10