mbin Documentation

Release 1.1.1

John Beaulaurier

Mar 20, 2018

Contents

mBin: a methylation-based binning framework for metagenomic SMRT sequencing reads

The mBin pipeline is designed to discover the unique signals of DNA methylation in metagenomic SMRT sequencing reads and leverage them for organism binning of assembled contigs or unassembled reads. Because *all* cellular DNA is modified by the same set of methyltransferases encoded in the genome, DNA methylation signals can be used for binning not just chromosomal sequences, but also extrachromosomal mobile genetic elements like plasmids.

The pipeline consists of four routines:

1. *buildcontrols*: Gets unmethylated IPD values for motifs from whole-genome amplified (WGA) sequencing
2. *filtermotifs*: Identifies methylated motifs in native metagenomic sequencing
3. *methylprofiles*: Creates methylation profiles for sequences using specified motifs
4. *mapfeatures*: Visualizes landscape of methylation features across all sequences

CHAPTER 1

Features

mBin can take as input either unaligned single molecule real-time (SMRT) reads from a PacBio instrument or contigs assembled from SMRT reads. Methylation scores are calculated from individual inter-pulse duration (IPD) metrics embedded in each sequencing read that record the polymerase kinetics during sequencing and indicate the presence or absence of DNA methylation at the level of individual nucleotides.

By aggregating these IPD metrics across multiple sites for a motif and across multiple reads aligned to a contig, mBin generates methylation scores for motifs and uses these to construct methylation profiles for reads and contigs. Methylation profiles can then be used as epigenetic features for unsupervised metagenomic binning. mBin can also generate methylation scores for contigs that are given binning assignments by other binning tools (with the –cross_cov_bins option).

CHAPTER 2

Installation

For a comprehensive guide on how to install mBin and its dependencies, see *Installation*

CHAPTER 3

Contribute

- Issue tracker: GitHub
- Source code: GitHub

CHAPTER 4

Contents

4.1 mBin overview

mBin: a methylation-based binning framework for metagenomic SMRT sequencing reads

The mBin pipeline is designed to discover the unique signals of DNA methylation in metagenomic SMRT sequencing reads and leverage them for organism binning of assembled contigs or unassembled reads. Because *all* cellular DNA is modified by the same set of methyltransferases encoded in the genome, DNA methylation signals can be used for binning not just chromosomal sequences, but also extrachromosomal mobile genetic elements like plasmids.

The pipeline consists of four routines:

1. *buildcontrols*: Gets unmethylated IPD values for motifs from whole-genome amplified (WGA) sequencing
2. *filtermotifs*: Identifies methylated motifs in native metagenomic sequencing
3. *methylprofiles*: Creates methylation profiles for sequences using specified motifs
4. *mapfeatures*: Visualizes landscape of methylation features across all sequences

4.1.1 Documentation

For a comprehensive guide on how to install and run mBin, please see the full documentation.

4.1.2 Citations

Beaulaurier J, Zhu S, Deikus G, Mogno I, Zhang XS, Davis-Richardson A, Canepa R, Triplett E, Faith J, Sebra R, Schadt EE, Fang G. Metagenomic binning and association of plasmids with bacterial host genomes using DNA methylation. *Nature Biotechnology* **36**, 61-69 (2018). doi:10.1038/nbt.4037.

4.1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

4.2 Installation

4.2.1 Fundamental dependencies

```
python v2.7.*
gcc
hdf5
```

4.2.2 Python packages

```
numpy>=1.7.1
pysam == 0.10.0
h5py >= 2.0.1
pbcore >= 0.9.4
scipy >= 0.12.0
biopython >= 1.6.1
matplotlib >= 1.5.0
```

All but Numpy will be installed automatically during the standard installation as described below. Numpy, however, must be installed prior to mBin installation (see below):

4.2.3 Setting up virtualenv

mBin should be installed in a Python virtual environment using virtualenv, which creates a clean and isolated Python environment in which to install packages and their dependencies.

Virtualenv can be installed using `pip`:

```
$ pip install virtualenv
```

Once installed, navigate to the directory where you would like to keep the virtual environment and create a virtual environment called `venv`:

```
$ virtualenv venv
```

Finally, activate this virtual environment `venv`:

```
$ . venv/bin/activate
```

Once activated, you are now operating inside the `venv` and should see the following on you command line:

```
(venv)<COMMAND LINE>
```

4.2.4 Installing mBin

With the virtual environment activated, install `mbin` using `pip`:

```
$ pip install mbin
```

4.2.5 Installing t-SNE

In order to create 2-D maps of methylation (and other) features for binning using *mapfeatures*, we must install the Barnes-Hut implementation of the t-SNE algorithm. Full details on the BH-tSNE algoritm and wrapper script can be found here. First, we pull the repository from GitHub and enter the directory:

```
$ git clone https://github.com/lvdmaaten/bhtsne.git
$ cd bhtsne
```

Next we compile the source code to get the executable `bh_tsne`:

```
$ g++ sptree.cpp tsne.cpp tsne_main.cpp -o bh_tsne -O2
```

Once the executable `bh_tsne` is compiled, add this directory to your `$PATH` environmental variable:

```
$ export PATH=$PATH:`pwd`
```

If `bh_tsne` is accessible in the path, the following should list usage instructions for *mapfeatures*:

```
$ mapfeatures --help
```

4.3 Usage

4.3.1 Extract control IPDs from WGA sequencing with *buildcontrols*

```
Usage: buildcontrols [--help] [options] wga_aligned_reads.cmp.h5

Example:

buildcontrols -i --procs=4 --control_pkl_name=control_means.pkl wga_aligned_reads.cmp.
↪h5

Options:
  -h, --help                              Show this help message and exit
  -d, --debug                             Increase verbosity of logging
  -i, --info                              Add basic logging
  --logFile=LOGFILE                       Write logging to file [log.controls]
  --subreadlength_min=SUBREADLENGTH_MIN   Minimum subread length to include for␣
↪analysis [100]
  --readlength_min=READLENGTH_MIN         Minimum read length to include for analysis␣
↪[100]
  --min_kmer=MIN_KMER                     Minimum motif size to scan (contiguous␣
↪motifs) [4]
  --max_kmer=MAX_KMER                     Maximum motif size to scan (contiguous␣
↪motifs) [6]
  --no_bipartite                          Omit bipartite motifs [False]
  --mod_bases=MOD_BASES                   String containing bases to query for mods.␣
↪Changing this is not recommended ['A']
  --minAcc=MINACC                         Min subread accuracy of read [0.8]
  --minMapQV=MINMAPQV                     Min mapping QV of aligned read [240]
  --procs=PROCS                           Number of cores to use [4]
  --N_reads=N_READS                       Number of qualifying reads to include in␣
↪analysis [1000000000]
```

(continues on next page)

(continued from previous page)

```
  --min_motif_count=MIN_MOTIF_COUNT       Number of motif sites required in WGA data␣
↪to be included in controls dictionary [10]
  --control_pkl_name=CONTROL_PKL_NAME     Filename to save control IPD data from WGA␣
↪sequencing [control_ipds.pkl]
```

All mBin analyses require a one-time step of creating a set of control IPD values using SMRT data from whole-genome amplified (WGA) sequencing. This WGA sequencing can be obtained from any bacterial genomic sequencing and does not have to be metagenomic.

The control (unmethylated) IPD values for all motifs that will be queried during the process of motif discovery and methylation profile construction. The IPD for each unmethylated motif is very dependant on the sequencing chemistry used and therefore best results are obtained by ensuring that the same chemistry kit (e.g. P6-C4) is used for both the WGA and native DNA sequencing runs.

This procedure collects control IPD information for all possible motifs in the defined query space as specified with the options `--min_kmer`, `--max_kmer`, `--bipart_first`, `--bipart_Ns`, and `--bipart_second`. Bipartite motifs describe those motifs (e.g. ACCTNNNNNCTT) that begin with a set of determinate bases (ACCT), followed by a set of indeterminate bases (NNNNN), followed by a second set of determinate bases (CTT). Accomodating every single possible bipartite motif configuration would cause the query motif space to balloon exponentially to a size that is not feasible for analysis. Therefore, constraints are placed on the acceptable lengths of each of these individual components of a bipartite motif using the options `--bipart_first`, `--bipart_Ns`, and `--bipart_second`.

This is a time- and resource-intensive process, but can be omitted for subsequent analyses once the control IPD values are extracted. However, the control IPD values should be re-extracted from WGA sequencing data whenever the sequencing chemistry is updated.

A WGA aligned_reads.cmp.h5 file containing SMRT read alignments is required for this step.

```
$ buildcontrols -i --procs=4 --control_pkl_name=control_means.pkl wga_aligned_reads.
↪cmp.h5
```

This process can be expedited in three ways:

1. Increasing `--procs`
2. Using `--N_reads` to only use a subset of the aligned reads in the WGA aligned_reads.cmp.h5 file, rather than using **all** aligned reads.
3. Using `--no_bipartite` to omit bipartite motifs (e.g. AGCNNNNNNGTCT) from the control dictionary, only creating control values for contiguous motifs (e.g. CTGCAG). However, this will serious hamper the ability to assess the full richness of methylated motifs in the native sequencing data.

The output of this step is a pickled file (called *control_means.pkl* in the above example) containing a dictionary of the mean IPD values for all queried motifs in the WGA data. These can be manually inspected in a Python interactive shell using the following commands:

```
>import pickle
>control_means = pickle.load(open("control_means.pkl", "rb"))
>for motif,ipd in control_means.iteritems():
>    print motif, ipd
```

Which should produce something similar to the following:

```
TAAGGA-5 0.22476
GGCAAG-4 -1.08934782609
ATANNNNNNTGCA-2 -0.306076923077
CTGATC-3 0.344641025641
ATANNNNNNTGCA-0 1.19992307692
```

(continues on next page)

(continued from previous page)

```
ATTCGG-0 0.000526315789474
GTCTA-4 0.151090909091
.....
```

It is important that the WGA sequencing data used for this step is of at least moderate depth and sequence complexity in order to provide sufficient control data points across the full spectrum of possible motifs. In subsequent analyses, any motifs lacking control IPD values will be discarded from the analysis, so try to include all motifs in the control data if possible.

4.3.2 Detect methylated motifs with *filtermotifs*

```
Usage: filtermotifs [--help] [options] aligned_reads.cmp.h5

Examples:

Using a cmp.h5 file of aligned reads as input (recommended):
filtermotifs -i --procs=4 --contigs=reference.fasta --control_pkl_name=control_means.
↪pkl aligned_reads.cmp.h5

Using a bas.h5 file of unaligned reads as input (not recommended):
filtermotifs -i --procs=4 --control_pkl_name=control_means.pkl m12345.bas.h5

Using a FOFN file of containing multiple files of bas.h5 unaligned reads as input␣
↪(not recommended):
filtermotifs -i --procs=4 --control_pkl_name=control_means.pkl bas.h5.fofn

Options:
  -h, --help                            Show this help message and exit
  -d, --debug                           Increase verbosity of logging
  -i, --info                            Add basic logging
  --logFile=LOGFILE                     Write logging to file [log.controls]
  --procs=PROCS                         Number of cores to use [4]
  --contigs=CONTIGS                     Fasta file containing entries for the␣
↪assembled contigs [None]
  --control_pkl_name=CONTROL_PKL_NAME    Filename of control IPD data from WGA␣
↪sequencing, generated using buildcontrols
                                        [control_ipds.pkl]
  --motifs_fn=MOTIFS_FN                 Filename to save output filtered motifs␣
↪[motifs.txt]
  --N_reads=N_READS                     Number of reads to include for motif␣
↪filtering [20000]
  --tmp=TMP                             Directory where numerous temporary files␣
↪will be written [filter_tmp]
  --minAcc=MINACC                       Min subread accuracy of read [0.8]
  --minMapQV=MINMAPQV                   Min mapping QV of aligned read [240]
  --minReadScore=MINREADSCORE           Min read score of an unaligned read [0.0]
  --maxPausiness=MAXPAUSINESS           Max pausiness value of an unaligned read␣
↪[1000]
  --subreadlength_min=SUBREADLENGTH_MIN  Minimum subread length to include for␣
↪analysis [100]
  --readlength_min=READLENGTH_MIN       Minimum read length to include for analysis␣
↪[100]
  --readlength_max=READLENGTH_MAX       Maximum read length to include for analysis␣
↪[10000000]
  --minQV=MINQV                         If base has QV < minQV, do not include [0]
```

(continues on next page)

(continued from previous page)

```
  --min_kmer=MIN_KMER                   Minimum motif size to scan (contiguous␣
↪motifs) [4]
  --max_kmer=MAX_KMER                   Maximum motif size to scan (contiguous␣
↪motifs) [6]
  --no_bipartite                        Omit bipartite motifs [False]
  --bipart_first=BIPART_FIRST           Bipartite motif configuration: acceptable␣
↪length of first determinate component
                                        (comma-separated string of integers) [3,4]
  --bipart_Ns=BIPART_NS                 Bipartite motif configuration: acceptable␣
↪length of middle indeterminate component
                                        (comma-separated string of integers) [5,6]
  --bipart_second=BIPART_SECOND         Bipartite motif configuration: acceptable␣
↪length of second determinate component
                                        (comma-separated string of integers) [3,4]
  --mod_bases=MOD_BASES                 String containing bases to query for mods ['A
↪']
  --minMotifIPD=MINMOTIFIPD             Min motif contig IPD for inclusion of motif␣
↪in final set [1.7]
  --min_motif_reads=MIN_MOTIF_READS     Min number of reads with motif hits to keep␣
↪for motif filtering (only if using
                                        unaligned reads as input) [20]
  --min_motif_N=MIN_MOTIF_N             Min number of motif IPD values required to␣
↪keep for motif filtering [20]
  --cross_cov_bins=CROSS_COV_BINS       Path to file containing binning results from␣
↪CONCOCT. Will use to improve motif
                                        discovery. Only works with contig-level␣
↪analysis (cmp.h5 input) inputs. File format
                                        should be '<contig_name>,<bin_id>' [None]
```

After the control IPD values have been tabulated and stored, methylated motifs can then detected in the HDF5 files of native, metagenomic sequencing data. Both aligned reads (cmp.h5) and unaligned reads (bas.h5 or FOFN containing multiple bas.h5 files) are supported as input, but the use of aligned reads is strongly recommended for motif filtering. Unaligned reads contain significant sequencing errors that introduce noise into the IPD signals for motifs, making it difficult to detect truly methylated motifs from unaligned reads.

```
filtermotifs -i --procs=4 --contigs=reference.fasta --control_pkl_name=control_means.
↪pkl native_aligned_reads.cmp.h5
```

Unless otherwise specified (using `--min_kmer`, `--max_kmer`, `--bipart_first`, `--bipart_Ns`, `--bipart_second`, or `--no_bipartite` options), this procedure starts with motifs for which control data exists (`--control_pkl_name`) and discards all motifs that do not have a methylation score (native IPD - control IPD) greater than the value defined by `--minMotifIPD`.

The original query space of motifs is very large and can include several hundred thousand motifs if bipartite motifs are included (recommended). To ease computational demands for storing IPD data for all motifs in the query space, only a subset of reads (`--N_reads`) are examined from the input HDF5 files. This value of `--N_reads` can be modified according to available computational resources and acceptable running time.

The filtered motifs are written to the output file specified by `--motifs_fn`. Motifs are listed with the sequence string, followed by the 0-based index of the methylated base. For example, GATC-1 indicates that the A position in the motif is methylated. This `--motifs_fn` output file serves as input to *methylprofiles*, which constructs methylation profiles across the filtered motifs.

4.3.3 Build methylation profiles with *methylprofiles*

```
Usage: methylprofiles [--help] [options] input_hdf5 motifs.txt

Example:

Using a cmp.h5 file of aligned reads as input:
methylprofiles -i --procs=4 --control_pkl_name=control_means.pkl --contigs=reference.
↪fasta aligned_reads.cmp.h5 motifs.txt

Using a bas.h5 file of unaligned reads as input:
methylprofiles -i --procs=4 --control_pkl_name=control_means.pkl m12345.bas.h5

Using a FOFN file of containing multiple files of bas.h5 unaligned reads as input:
methylprofiles -i --procs=4 --control_pkl_name=control_means.pkl bas.h5.fofn

Options:
  -h, --help                                show this help message and exit
  -d, --debug                               Increase verbosity of logging
  -i, --info                                Add basic logging
  --logFile=LOGFILE                         Write logging to file [log.controls]
  --prefix=PREFIX                           Prefix to use for output files [None]
  --tmp=TMP                                 Directory where numerous temporary files␣
↪will be written [profiles_tmp]
  --contigs=CONTIGS                         Fasta file containing entries for the␣
↪assembled contigs [None]
  --minReadScore=MINREADSCORE               Min read score of an unaligned read [0.0]
  --maxPausiness=MAXPAUSINESS               Max pausiness value of an unaligned read␣
↪[1000]
  --minQV=MINQV                             If base has QV < minQV, do not include [0]
  --subreadlength_min=SUBREADLENGTH_MIN     Minimum subread length to include for␣
↪analysis [100]
  --readlength_min=READLENGTH_MIN           Minimum read length to include for analysis␣
↪[100]
  --minContigLength=MINCONTIGLENGTH         Min length of contig to consider [10000]
  --comp_kmer=COMP_KMER                     Kmer size to use for sequence composition␣
↪measurements [5]
  --aligned_read_barcodes                   Also output features for individual aligned␣
↪reads, not just
                                            contigs (requires cmp.h5 input) [False]
  --minAcc=MINACC                           Min subread accuracy of read [0.8]
  --minMapQV=MINMAPQV                       Min mapping QV of aligned read [240]
  --procs=PROCS                             Number of processors to use [4]
  --N_reads=N_READS                         Number of qualifying reads to include (from␣
↪each bas.h5 if input is FOFN
                                            of bas.h5 files) in analysis [1000000000]
  --control_pkl_name=CONTROL_PKL_NAME       Filename to save control IPD data from WGA␣
↪sequencing [control_ipds.pkl]
  --subtract_control=SUBTRACT_CONTROL       Subtract control IPDs in final calculations␣
↪[True]
  --cross_cov_bins=CROSS_COV_BINS           Path to file containing binning results␣
↪from CONCOCT. Will use to improve
                                            motif discovery. Only works with contig-
↪level analysis (cmp.h5 input) inputs.
                                            File format should be '<contig_name>,<bin_
↪id>' [None]
```

methylprofiles compiles the methylation profiles across the motifs specified in the arguments. The methylation profiles

can be constructed using either native contigs (*.cmp.h5) or unaligned reads (*.bas.h5), the latter of which can be supplied as a single bas.h5 file or a FOFN containing multiple *.bas.h5 files (each *bas.h5 file on a new line).

The output consists of three separate files containing methylation features, as well as other relevant features for binning:

1. **Methylation features: <prefix>_<seq>_methyl_features.txt**
    - Column 1: <seq> id
    - Column 2: <seq> length
    - Columns 3-M: Methylation scores (for M motifs)
2. **Motif counts: <prefix>_<seq>_motif_counts.txt**
    - Column 1: <seq> id
    - Column 2: <seq> length
    - Columns 3-M: Motif counts (for M motifs)
3. **Alternative features: <prefix>_<seq>_other_features.txt**
    - Column 1: <seq> id
    - Column 2: <seq> length
    - **For dtype = read or align:**
        - Columns 3-N: k-mer frequencies (for N k-mers)
    - **For dtype = contig:**
        - Column 3: Contig coverage
        - Columns 4-N: k-mer frequencies (for N k-mers)

Where <prefix> is defined by `--prefix` and <seq> is the sequence data type: contig, align, or read. When inputting *.cmp.h5 reads for methylation profiling, *methylprofiles* will always generate contig level features and will optionally generate align level features for the reads comprising each contig (using `--aligned_read_barcodes`). When *.bas.h5 reads are input, only read level features will be output.

For example, the following command with generate methylation (and other) profiles for a set of contigs contained in aligned_reads.cmp.h5:

```
methylprofiles -i --procs=4 --prefix=test --control_pkl_name=control_means.pkl --
↪contigs=reference.fasta aligned_reads.cmp.h5 motifs.txt
```

These profiles will be contained in the following output files:

1. test_contig_methyl_features.txt
2. test_contig_motif_counts.txt
3. test_contig_other_features.txt

4.3.4 Visualize feature landscape with *mapfeatures*

```
Usage: mapfeatures [--help] [options] <SEQ>_methyl_features.txt <SEQ>_other_features.
↪txt

Examples:
```

(continues on next page)

(continued from previous page)

```
mapfeatures -i --labels=<LABELS.txt> --size_markers contig_methyl_features.txt contig_
↪other_features.txt

mapfeatures -i --labels=<LABELS.txt> --l_min=500 align_methyl_features.txt align_
↪other_features.txt

mapfeatures -i --l_min=10000 --n_seqs=1000 read_methyl_features.txt read_other_
↪features.txt

Options:
  -h, --help                Show this help message and exit
  -d, --debug               Increase verbosity of logging
  -i, --info                Add basic logging
  --logFile=LOGFILE         Write logging to file [log.controls]
  --prefix=PREFIX           Prefix to use for output files [None]
  --size_markers            Adjust marker size in plot according to sequence length␣
↪[False]
  --dim_reduce=DIM_REDUCE   Dimensionality reduction algorithm to apply (bhtsne or␣
↪pca) [bhtsne]
  --labels=LABELS           Tab-delimited file (no header) of sequence labels (seq_
↪name\tlabel_name) [None]
  --l_min=L_MIN             Minimum read length to include for analysis [0]
  --n_seqs=N_SEQS           Number of sequences to subsample [all]
  --n_dims=N_DIMS           Number of dimensions to reduce to for visualization (only␣
↪n_dims=2 will be plotted) [2]
  --n_iters=N_ITERS         Number of iterations to use for BH-tSNE [500]
```

mapfeatures visualizes the landscape of high-dimensional sequence features using the Barnes Hut approximation of t-SNE (*PCA support coming soon*). The sequence features that are output from *methylprofiles* are often high-dimensional (>3D), making it difficult to visualize the sequences. To ease this visualization for resolution of discrete sequence clusters in the feature space, t-SNE is used to reduce the dimensionality of the methylation, composition, and coverage features to 2D. The resulting 2D maps, which can be overlaid with sequence annotation labels (generated with Kraken, for instance), often reveals sequence clustering in the 2D feature space representing distinct taxonomical groups for binning.

Two files from *methylprofiles* serve as input to *mapfeatures*: `<prefix>_<seq>_methyl_features.txt` and `<prefix>_<seq>_other_features.txt`, which contain the high-dimensional methylation score features and coverage & composition features, respectively. As before, <prefix> is defined by `--prefix` and <seq> is the sequence data type: contig, align, or read.

For example, the command:

```
mapfeatures -i --labels=<LABELS.txt> --size_markers contig_methyl_features.txt contig_
↪other_features.txt
```

Will generate three 2D t-SNE maps representing the contig feature space:

1. **Methylation t-SNE map**: contig_methyl_features.bhtsne.png
2. **Composition t-SNE map**: contig_other_features.comp.bhtsne.png
3. **Coverage+composition t-SNE map**: contig_other_features.covcomp.bhtsne.png

Each file has an accompanying tab-delimited *.txt file containing the 2D coordinates from each t-SNE map that can be used for additional plotting purposes by the user. The file supplied to `--labels` must contain tab-delimited sequence labels for **all** sequences listed in the first column of the <seq>_methyl_features.txt and <seq>_other_features.txt files that are output from *methylprofiles*. Any sequences lacking annotation must be included using the label 'Unlabeled', for example:

contig_1	Bacteroides
contig_2	Clostridium
contig_4	Escherichia
contig_5	Bacteroides
contig_6	Unlabeled

4.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/fanglab/mbin/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

mbin could always use more documentation, whether as part of the official mbin docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/fanglab/mbin/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.4.2 Get Started!

Ready to contribute? Here's how to set up *mbin* for local development.

1. Fork the *mbin* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mbin.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mbin
$ cd mbin/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mbin tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/fanglab/mbin/pull_requests and make sure that the tests pass for all supported Python versions.

4.4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_mbin
```

CHAPTER 5

Search

- search