Autometa

Release 2.0

Ian J. Miller, Evan R. Rees, Izaak Miller, Shaurya Chanana, Siddha

Jan 29, 2024

CONTENTS

1 Guide 3

Python Module Index 151

Index 153

Autometa: automated extraction of microbial genomes from individual shotgun metagenomes

An automated binning pipeline for single metagenomes, in particular host-associated and highly complex ones.

If you find Autometa useful to your work, please cite our Autometa paper:

Miller, I. J.; Rees, E. R.; Ross, J.; Miller, I.; Baxa, J.; Lopera, J.; Kerby, R. L.; Rey, F. E.; Kwan, J. C. Autometa: Automated extraction of microbial genomes from individual shotgun metagenomes. *Nucleic Acids Research*, **2019**. DOI: https://doi.org/10.1093/nar/gkz148

CHAPTER

ONE

GUIDE

1.1 Getting Started

You will need to specifically configure your compute environment depending on how you would like to run the Autometa workflow.

1.1.1 Choose a workflow

- *Nextflow Workflow*
- *Bash Workflow*

1.2 Nextflow Workflow

1.2.1 Why nextflow?

Nextflow helps Autometa produce reproducible results while allowing the pipeline to scale across different platforms and hardware.

1.2.2 System Requirements

Currently the nextflow pipeline requires Docker so it must be installed on your system. If you don't have Docker installed you can install it from docs.docker.com/get-docker. We plan on removing this dependency in future versions, so that other dependency managers (e.g. Conda, Mamba, Singularity, etc) can be used.

Nextflow runs on any Posix compatible system. Detailed system requirements can be found in the nextflow documentation

Nextflow (required) and nf-core tools (optional but highly recommended) installation will be discussed in *Installing Nextflow and nf-core tools with mamba*.

1.2.3 Data Preparation

1. *Metagenome Assembly*
2. *Preparing a Sample Sheet*

Metagenome Assembly

You will first need to assemble your shotgun metagenome, to provide to Autometa as input.

The following is a typical workflow for metagenome assembly:

1. Trim adapter sequences from the reads
    - We usually use Trimmomatic.
2. Quality check the trimmed reads to ensure the adapters have been removed
    - We usually use FastQC.
3. Assemble the trimmed reads
    - We usually use MetaSPAdes which is a part of the SPAdes package.
4. Check the quality of your assembly (Optional)
    - We usually use metaQuast for this (use `--min-contig 1` option to get an accurate N50).

      This tool can compute a variety of assembly statistics one of which is N50. This can often be useful for selecting an appropriate length cutoff value for pre-processing the metagenome.

Preparing a Sample Sheet

An example sample sheet for three possible ways to provide a sample as an input is provided below. The first example provides a metagenome with paired-end read information, such that contig coverages may be determined using a read-based alignment sub-workflow. The second example uses pre-calculated coverage information by providing a coverage table *with* the input metagenome assembly. The third example retrieves coverage information from the assembly contig headers (Currently, this is only available with metagenomes assembled using SPAdes)

Note: If you have paired-end read information, you can supply these file paths within the sample sheet and the coverage table will be computed for you (See `example_1` in the example sheet below).

If you have used any other assembler, then you may also provide a coverage table (See `example_2` in the example sheet below). Fortunately, Autometa can construct this table for you with: `autometa-coverage`. Use `--help` to get the complete usage or for a few examples see *2. Coverage calculation*.

If you use SPAdes then Autometa can use the k-mer coverage information in the contig names (`example_3` in the example sample sheet below).

sample	assembly	fastq_1	fastq_2	coverage_tab	cov_from_assembly
example_1	/path/to/example/1/meta	/path/to/paired-end/fwd_reads.fastq.gz	/path/to/paired-end/rev_reads.fastq.gz		0
example_2	/path/to/example/2/meta			/path/to/cover	0
example_3	/path/to/example/3/meta				spades

Note: To retrieve coverage information from a sample's contig headers, provide the `assembler` used for the sample value in the row under the `cov_from_assembly` column. Using a `0` will designate to the workflow to try to retrieve coverage information from the coverage table (if it is provided) or coverage information will be calculated by read alignments using the provided paired-end reads. If both paired-end reads and a coverage table are provided, the pipeline will prioritize the coverage table.

If you are providing a coverage table to `coverage_tab` with your input metagenome, it must be tab-delimited and contain *at least* the header columns, `contig` and `coverage`.

Supported Assemblers for `cov_from_assembly`

Assembler	Supported (Y/N)	`cov_from_assembly`
[meta]SPAdes	Y	`spades`
Unicycler	N	`unicycler`
Megahit	N	`megahit`

You may copy the below table as a csv and paste it into a file to begin your sample sheet. You will need to update your input parameters, accordingly.

Example `sample_sheet.csv`

```
sample,assembly,fastq_1,fastq_2,coverage_tab,cov_from_assembly
example_1,/path/to/example/1/metagenome.fna.gz,/path/to/paired-end/fwd_reads.fastq.gz,/
↪path/to/paired-end/rev_reads.fastq.gz,,0
example_2,/path/to/example/2/metagenome.fna.gz,,,/path/to/coverage.tsv,0
example_3,/path/to/example/3/metagenome.fna.gz,,,,spades
```

Caution: Paths to any of the file inputs **must be absolute file paths**.

Incorrect	Correct	Description
`$HOME/Autometa/tests/data/metagenome.fna.gz`	`/home/user/Autometa/tests/data/metagenome.fna.gz`	Replacing any instance of the `$HOME` variable with the real path
`tests/data/metagenome.fna.gz`	`/home/user/Autometa/tests/data/metagenome.fna.gz`	Using the entire file path of the input

1.2.4 Quick Start

The following is a condensed summary of steps required to get Autometa installed, configured and running. There are links throughout to the appropriate documentation sections that can provide more detail if required.

Installation

For full installation instructions, please see the *Installation* section

If you would like to install Autometa via mamba (I'd recommend it, its almost foolproof!), you'll need to first download the Mambaforge installer on your system. You can do this in a few easy steps:

1. Type in the following and then hit enter. This will download the Mambaforge installer to your home directory.

```
wget "https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-
↪$(uname)-$(uname -m).sh" -O "$HOME/Mambaforge-$(uname)-$(uname -m).sh"
```

Note: `$HOME` is synonymous with `/home/user` and in my case is `/home/sam`

2. Now let's run the installer. Type in the following and hit enter:

```
bash $HOME/Mambaforge-$(uname)-$(uname -m).sh
# On my machine this was /home/sam/Mambaforge-latest-Linux-x86_64.sh
```

3. Follow all of the prompts. Keep pressing enter until it asks you to accept. Then type yes and enter. Say yes to everything.

Note: If for whatever reason, you accidentally said no to the initialization, do not fear. We can fix this by running the initialization with the following command:

```
$HOME/mambaforge/bin/mamba init
```

1. Finally, for the changes to take effect, you'll need to run the following line of code which effectively acts as a "refresh"

```
source $HOME/.bashrc
```

Now that you have mamba up and running, its time to install the Autometa mamba environment. Run the following code:

```
mamba env create --file=https://raw.githubusercontent.com/KwanLab/Autometa/main/nextflow-
↪env.yml
```

Attention: You will only need to run the installation (code above) once. The installation does NOT need to be performed every time you wish to use Autometa. Once installation is complete, the mamba environment (which holds all the tools that Autometa needs) will live on your server/computer much like any other program you install.

Anytime you would like to run Autometa, you'll need to activate the mamba environment. To activate the environment you'll need to run the following command:

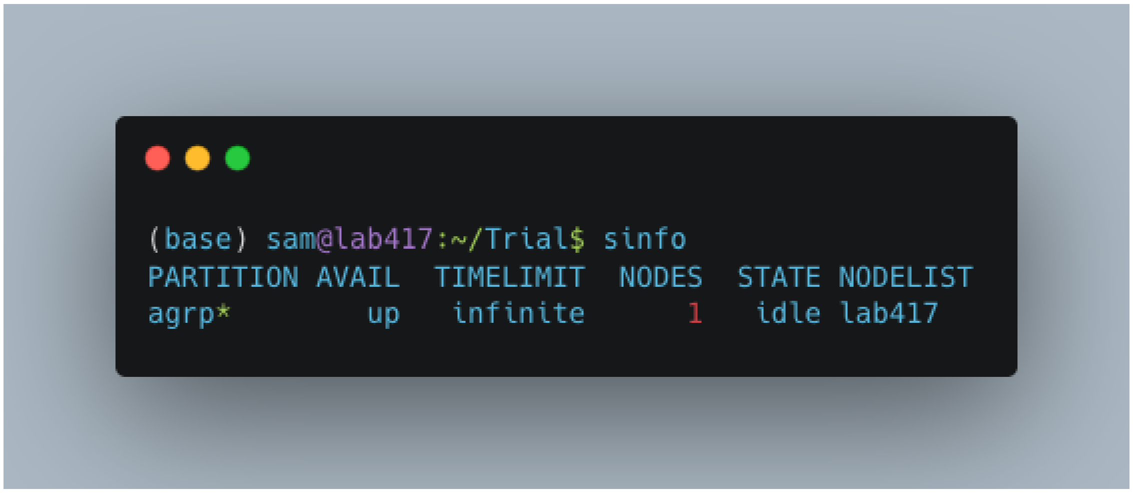
```
mamba activate autometa-nf
```

Configuring a scheduler

For full details on how to configure your scheduler, please see the *Configuring your process executor* section.

If you are using a Slurm scheduler, you will need to create a configuration file. If you do not have a scheduler, skip ahead to *Running Autometa*

First you will need to know the name of your slurm partition. Run `sinfo` to find this. In the example below, the partition name is “agrp”.



```
(base) sam@lab417:~/Trial$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
agrp*        up   infinite      1   idle lab417
```

Next, generate a new file called `slurm_nextflow.config` via nano:

```
nano slurm_nextflow.config
```

Then copy the following code block into that new file (“agrp” is the slurm partition to use in our case):

```
profiles {
        slurm {
            process.executor       = "slurm"
            process.queue          = "agrp" // <<-- change this to whatever your␣
↪partition is called
            docker.enabled         = true
            docker.userEmulation   = true
            singularity.enabled    = false
            podman.enabled         = false
            shifter.enabled        = false
            charliecloud.enabled   = false
            executor {
                queueSize = 8
            }
        }
    }
```

Keep this file somewhere central to you. For the sake of this example I will be keeping it in a folder called “Useful scripts” in my home directory because that is a central point for me where I know I can easily find the file and it won’t be moved e.g. `/home/sam/Useful_scripts/slurm_nextflow.config`

Save your new file with Ctrl+O and then exit nano with Ctrl+O.

Installation and set up is now complete.

Running Autometa

For a comprehensive list of features and options and how to use them please see *Running the pipeline*

Autometa can bin one or several metagenomic datasets in one run. Regardless of the number of metagenomes you want to process, you will need to provide a sample sheet which specifies the name of your sample, the full path to where that data is found and how to retrieve the sample's contig coverage information.

If the metagenome was assembled via SPAdes, Autometa can extract coverage and contig length information from the sequence headers.

If you used a different assembler you will need to provide either raw reads or a table containing contig/scaffold coverage information. Full details for data preparation may be found under *Preparing a Sample Sheet*

First ensure that your Autometa mamba environment is activated. You can activate your environment by running:

```
mamba activate autometa-nf
```

Run the following code to launch Autometa:

```
nf-core launch KwanLab/Autometa
```

Note: You may want to note where you have saved your input sample sheet prior to running the launch command. It is much easier (and less error prone) to copy/paste the sample sheet file path when specifying the input (We'll get to this later in *Input and Output*).

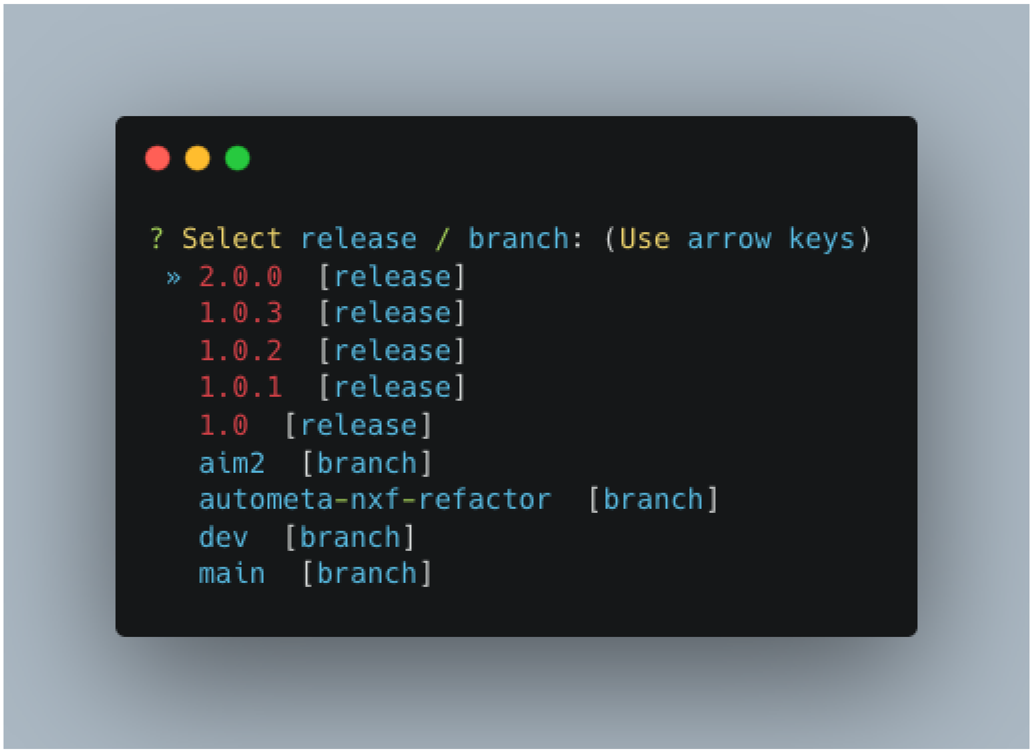
You will now use the arrow keys to move up and down between your options and hit your "Enter" or "Return" key to make your choice.

KwanLab/Autometa nf-core parameter settings:

1. *Choose a version*
2. *Choose nf-core interface*
3. *General nextflow parameters*
4. *Input and Output*
5. *Binning parameters*
6. *Additional Autometa options*
7. *Computational parameters*
8. *Do you want to run the nextflow command now?*

Choose a version

The double, right-handed arrows should already indicate the latest release of Autometa (in our case `2.0.0`). The latest version of the tool will always be at the top of the list with older versions descending below. To select the latest version, ensure that the double, right-handed arrows are next to `2.0.0`, then hit "Enter".

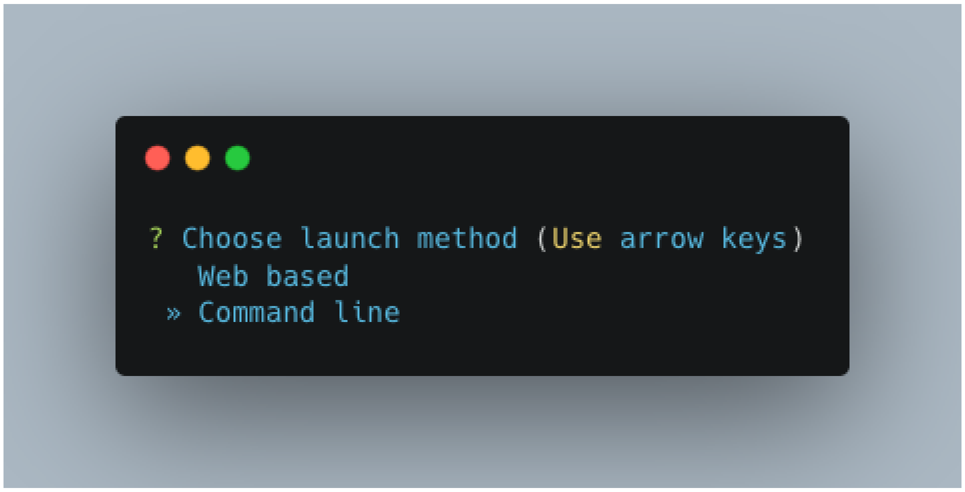


```
? Select release / branch: (Use arrow keys)
 » 2.0.0  [release]
   1.0.3  [release]
   1.0.2  [release]
   1.0.1  [release]
   1.0  [release]
   aim2  [branch]
   autometa-nxf-refactor  [branch]
   dev  [branch]
   main  [branch]
```

Choose nf-core interface

Pick the `Command line` option.

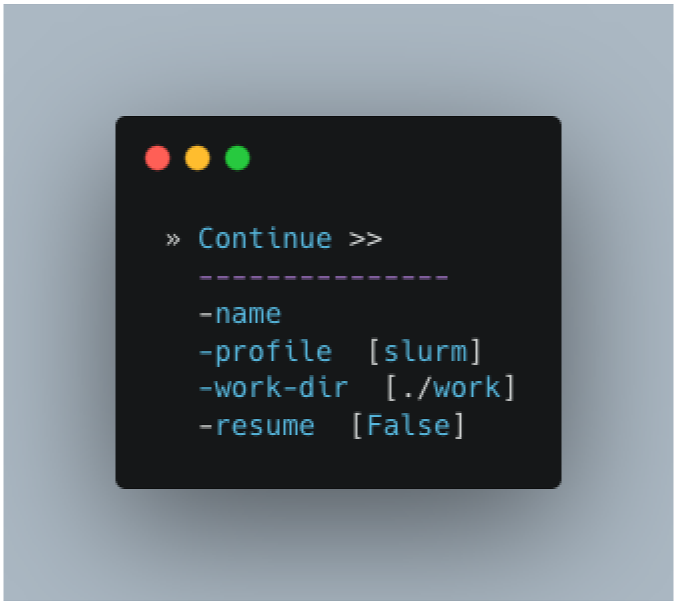
Note: Unless you've done some fancy server networking (i.e. tunneling and port-forwarding), or are using Autometa locally, `Command line` is your *only* option.



```
? Choose launch method (Use arrow keys)
   Web based
 » Command line
```

General nextflow parameters

If you are using a scheduler (Slurm in this example), `-profile` is the only option you'll need to change. If you are not using a scheduler, you may skip this step.

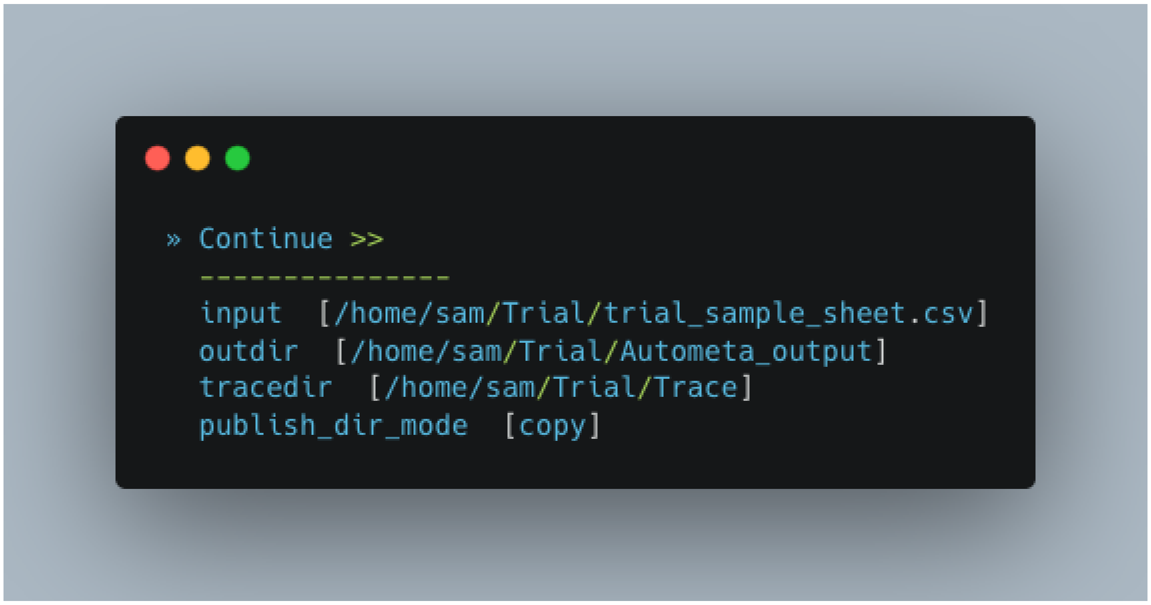


```
 » Continue >>
   ---------------
   -name
   -profile  [slurm]
   -work-dir  [./work]
   -resume  [False]
```

Input and Output

Now we need to give Autometa the full paths to our input sample sheet, output results folder and output logs folder (aka where trace files are stored).

Note: A new folder, named by its respective sample value, will be created within the output results folder for each metagenome listed in the sample sheet.

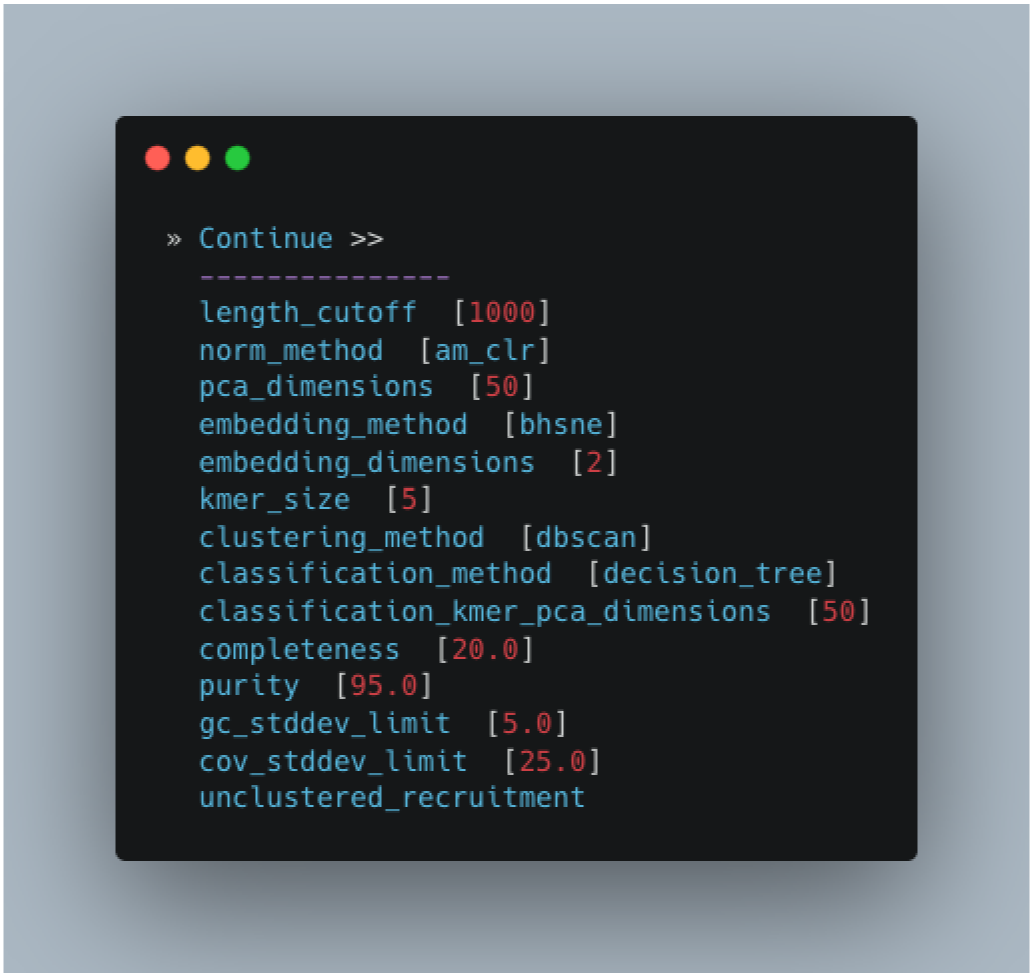


```
» Continue >>
  ---------------
  input  [/home/sam/Trial/trial_sample_sheet.csv]
  outdir  [/home/sam/Trial/Autometa_output]
  tracedir  [/home/sam/Trial/Trace]
  publish_dir_mode  [copy]
```

Binning parameters

If you're not sure what you're doing I would recommend only changing `length_cutoff`. The default cutoff is 3000bp, which means that any contigs/scaffolds smaller than 3000bp will not be considered for binning.

Note: This cutoff will depend on how good your assembly is: e.g. if your N50 is 1200bp, I would choose a cutoff of 1000. If your N50 is more along the lines of 5000, I would leave the cutoff at the default 3000. I would strongly recommend against choosing a number below 900 here. In the example below, I have chosen a cutoff of 1000bp as my assembly was not particularly great (the N50 is 1100bp).



```
» Continue >>
  ---------------
  length_cutoff  [1000]
  norm_method  [am_clr]
  pca_dimensions  [50]
  embedding_method  [bhsne]
  embedding_dimensions  [2]
  kmer_size  [5]
  clustering_method  [dbscan]
  classification_method  [decision_tree]
  classification_kmer_pca_dimensions  [50]
  completeness  [20.0]
  purity  [95.0]
  gc_stddev_limit  [5.0]
  cov_stddev_limit  [25.0]
  unclustered_recruitment
```

Additional Autometa options

Here you have a choice to make:

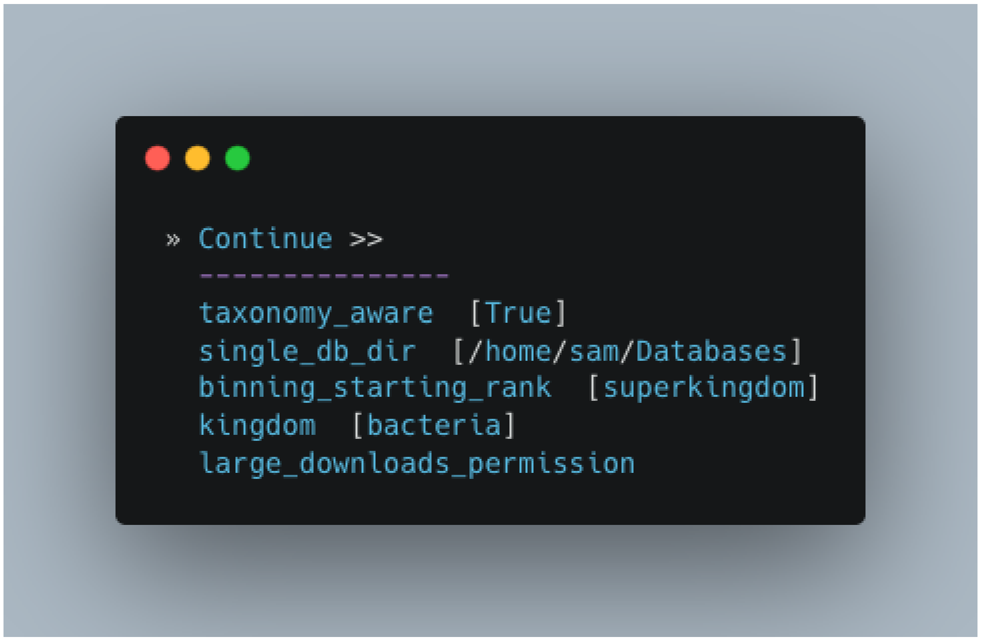
- By enabling taxonomy aware mode, Autometa will attempt to use taxonomic data to make your bins more accurate.

However, this is a more computationally expensive step and will make the process take longer.

- By leaving this option as the default `False` option, Autometa will bin according to coverage and kmer patterns.

Despite your choice, you will need to provide a path to the necessary databases using the `single_db_dir` option. In the example below, I have enabled the taxonomy aware mode and provided the path to where the databases are stored (in my case this is `/home/sam/Databases`).

For additional details on required databases, see the *Databases* section.



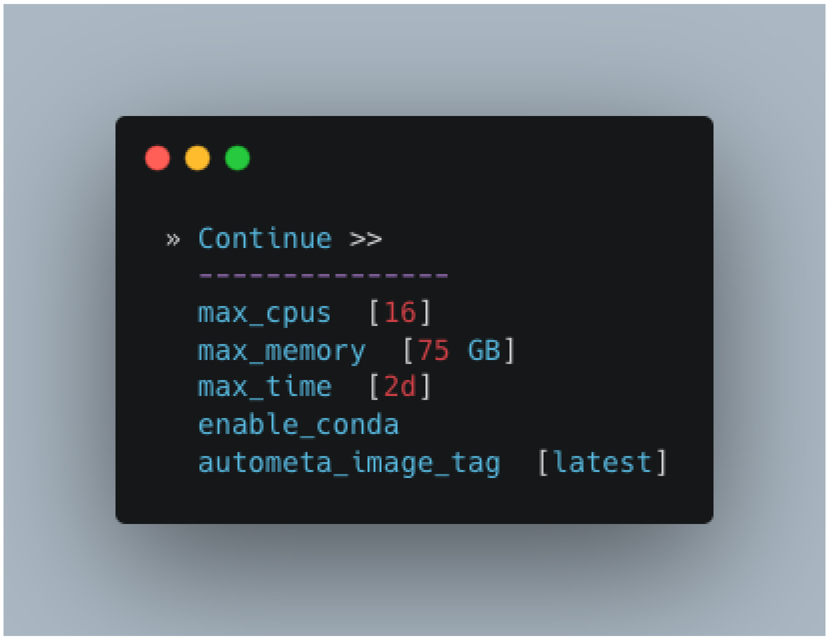
```
» Continue >>
  ---------------
  taxonomy_aware  [True]
  single_db_dir  [/home/sam/Databases]
  binning_starting_rank  [superkingdom]
  kingdom  [bacteria]
  large_downloads_permission
```

Computational parameters

This will depend on the computational resources you have available. You could start with the default values and see how the binning goes. If you have particularly complex datasets you may want to bump this up a bit. For your average metagenome, you won't need more than 150Gb of memory. I've opted to use 75 Gb as a starting point for a few biocrust (somewhat diverse) metagenomes.

Note: These options correspond to the resources provided to *each* process of Autometa, *not* the entire workflow itself.

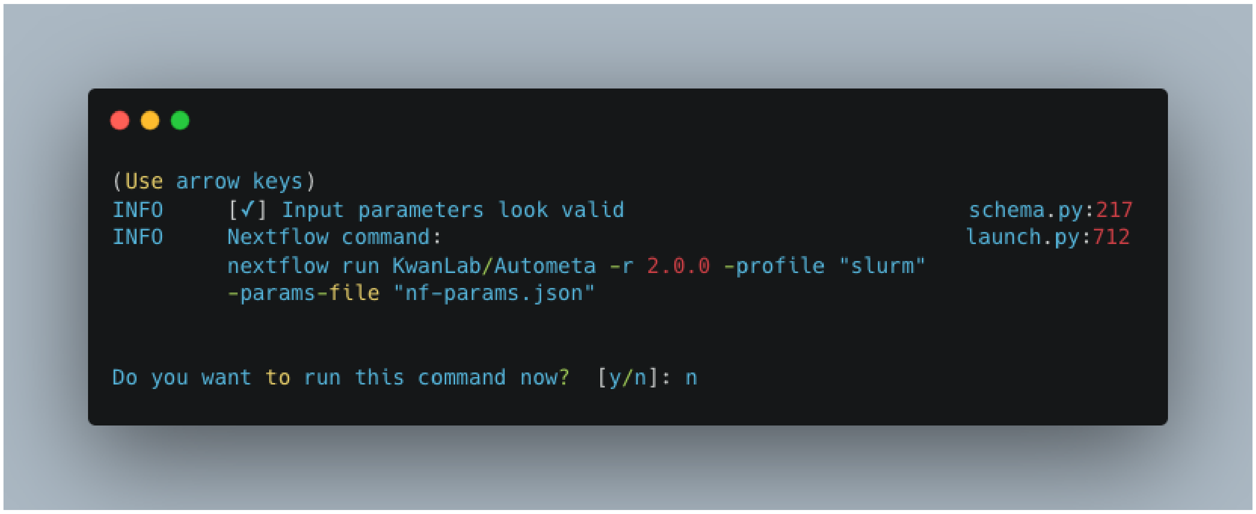
Also, for TB worth of assembled data you may want to try the *Bash Workflow* using the autometa-large-data-mode.sh template



```
» Continue >>
  ---------------
  max_cpus  [16]
  max_memory  [75 GB]
  max_time  [2d]
  enable_conda
  autometa_image_tag  [latest]
```

Do you want to run the nextflow command now?

You will now be presented with a choice. If you are NOT using a scheduler, you can go ahead and type `y` to launch the workflow. If you are using a scheduler, type `n` - we have one more step to go. In the example below, I am using the slurm scheduler so I have typed `n` to prevent immediately performing the nextflow run command.



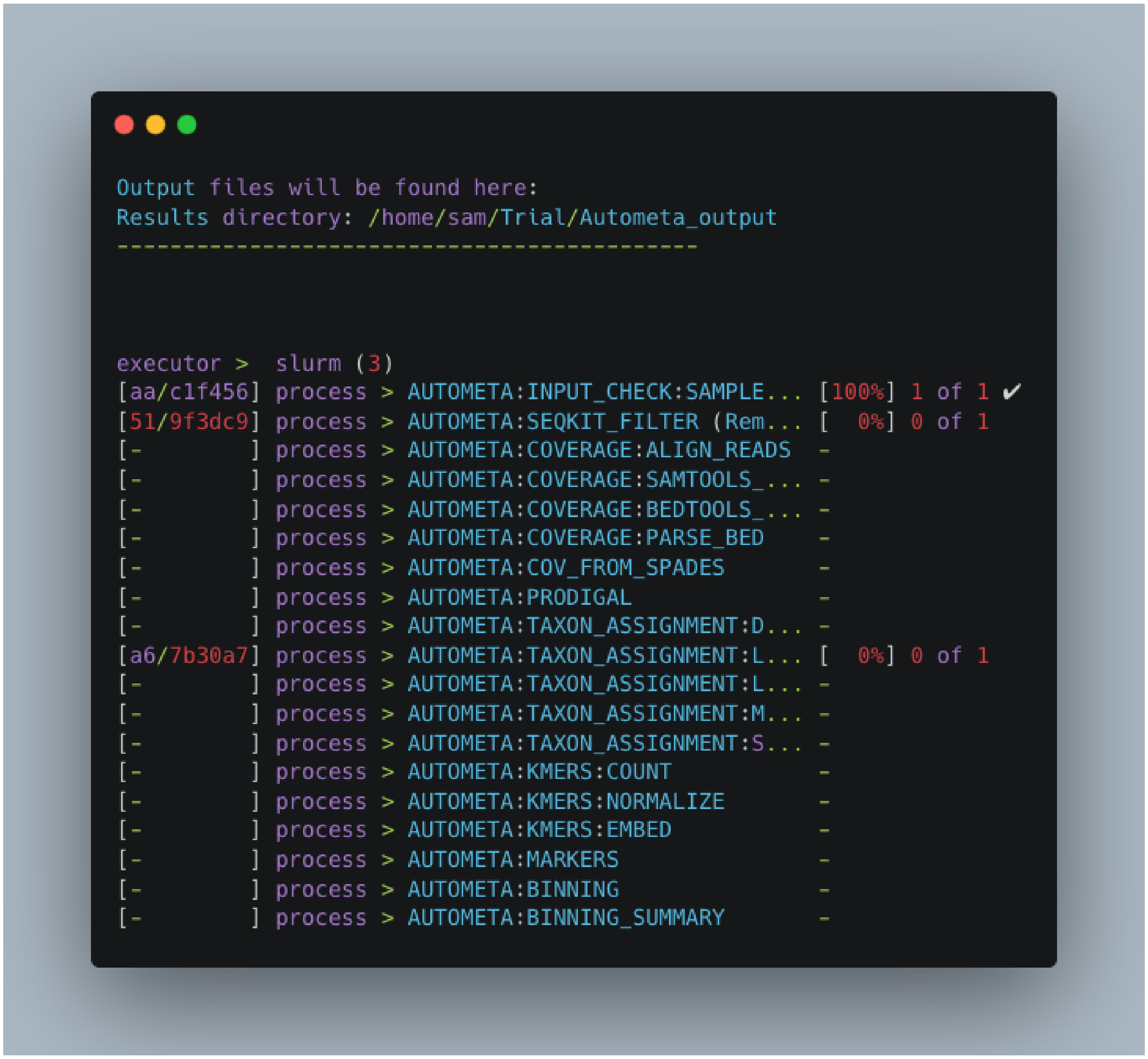
```
(Use arrow keys)
INFO     [✓] Input parameters look valid                        schema.py:217
INFO     Nextflow command:                                       launch.py:712
         nextflow run KwanLab/Autometa -r 2.0.0 -profile "slurm"
         -params-file "nf-params.json"


Do you want to run this command now?  [y/n]: n
```

If you recall, we created a file called `slurm_nextflow.config` that contains the information Autometa will need to communicate with the Slurm scheduler. We need to include that file using the `-c` flag (or configuration flag). Therefore to launch the Autometa workflow, run the following command:

Note: You will need to change the `/home/sam/Useful_scripts/slurm_nextflow.config` file path to what is appropriate for your system.

```
nextflow run KwanLab/Autometa -r 2.0.0 -profile "slurm" -params-file "nf-params.json" -c
↪"/home/sam/Useful_scripts/slurm_nextflow.config"
```

Once you have hit the "Enter" key to submit the command, nextflow will display the progress of your binning run, such as the one below:



```
Output files will be found here:
Results directory: /home/sam/Trial/Autometa_output
--------------------------------------------



executor >  slurm (3)
[aa/c1f456] process > AUTOMETA:INPUT_CHECK:SAMPLE... [100%] 1 of 1 ✔
[51/9f3dc9] process > AUTOMETA:SEQKIT_FILTER (Rem... [  0%] 0 of 1
[-        ] process > AUTOMETA:COVERAGE:ALIGN_READS  -
[-        ] process > AUTOMETA:COVERAGE:SAMTOOLS_... -
[-        ] process > AUTOMETA:COVERAGE:BEDTOOLS_... -
[-        ] process > AUTOMETA:COVERAGE:PARSE_BED    -
[-        ] process > AUTOMETA:COV_FROM_SPADES       -
[-        ] process > AUTOMETA:PRODIGAL              -
[-        ] process > AUTOMETA:TAXON_ASSIGNMENT:D... -
[a6/7b30a7] process > AUTOMETA:TAXON_ASSIGNMENT:L... [  0%] 0 of 1
[-        ] process > AUTOMETA:TAXON_ASSIGNMENT:L... -
[-        ] process > AUTOMETA:TAXON_ASSIGNMENT:M... -
[-        ] process > AUTOMETA:TAXON_ASSIGNMENT:S... -
[-        ] process > AUTOMETA:KMERS:COUNT           -
[-        ] process > AUTOMETA:KMERS:NORMALIZE       -
[-        ] process > AUTOMETA:KMERS:EMBED           -
[-        ] process > AUTOMETA:MARKERS               -
[-        ] process > AUTOMETA:BINNING               -
[-        ] process > AUTOMETA:BINNING_SUMMARY       -
```

When the run is complete, output will be stored in your designated output folder, in my case `/home/same/Trial/Autometa_output` (See *Input and Output*).

1.2.5 Basic

While the Autometa Nextflow pipeline can be run using Nextflow directly, we designed it using nf-core standards and templating to provide an easier user experience through use of the nf-core “tools” python library. The directions below demonstrate using a minimal mamba environment to install Nextflow and nf-core tools and then running the Autometa pipeline.

Installing Nextflow and nf-core tools with mamba

If you have not previously installed/used mamba, you can get it from Mambaforge.

After installing mamba, running the following command will create a minimal mamba environment named "autometa-nf", and install Nextflow and nf-core tools.

```
mamba env create --file=https://raw.githubusercontent.com/KwanLab/Autometa/main/nextflow-
↪env.yml
```

If you receive the message…

```
CondaValueError: prefix already exists: /home/user/mambaforge/envs/autometa-nf
```

…it means you have already created the environment. If you want to overwrite/update the environment then add the `--force` flag to the end of the command.

```
mamba env create --file=https://raw.githubusercontent.com/KwanLab/Autometa/main/nextflow-
↪env.yml --force
```

Once mamba has finished creating the environment be sure to activate it:

```
mamba activate autometa-nf
```

Using nf-core

Download/Launch the Autometa Nextflow pipeline using nf-core tools. The stable version of Autometa will always be the "main" git branch. To use an in-development git branch switch "main" in the command with the name of the desired branch. After the pipeline downloads, nf-core will start the pipeline launch process.

```
nf-core launch KwanLab/Autometa
```

Caution: nf-core will give a list of revisions to use following the above command. Any of the version 1.* revisions are NOT supported.

Attention: If you receive an error about schema parameters you may be able to resolve this by first removing the existing project and pulling the desired `KwanLab/Autometa` project using `nextflow`.

If a local project exists (you can check with `nextflow list`), first `drop` this project:

`nextflow drop KwanLab/Autometa`

Now pull the desired revision:

```
nextflow pull KwanLab/Autometa -r 2.0.0
# or
nextflow pull KwanLab/Autometa -r main
# or
nextflow pull KwanLab/Autometa -r dev
# Now run nf-core with selected revision from above
nf-core launch KwanLab/Autometa -r <2.0.0|main|dev>
```

Now after re-running `nf-core launch ...` select the revision that you downloaded from above.

You will then be asked to choose "Web based" or "Command line" for selecting/providing options. While it is possible to use the command line version, it is preferred and easier to use the web-based GUI. Use the arrow keys to select one or the other and then press return/enter.

Setting parameters with a web-based GUI

The GUI will present all available parameters, though some extra parameters may be hidden (these can be revealed by selecting "Show hidden params" on the right side of the page).

Required parameters

The first required parameter is the input sample sheet for the Autometa workflow, specified using `--input`. This is the path to your input sample sheet. See *Preparing a Sample Sheet* for additional details.

The other parameter is a nextflow argument, specified with `-profile`. This configures nextflow and the Autometa workflow as outlined in the respective "profiles" section in the pipeline's `nextflow.config` file.

- `standard` (default): runs all process jobs locally, (currently this requires Docker, i.e. docker is enabled for all processes the default profile).
- `slurm`: submits all process jobs into the slurm queue. See *SLURM* before using
- `docker`: enables docker for all processes

Caution: Additional profiles exists in the `nextflow.config` file, however these have not yet been tested. If you are able to successfully configure these profiles, please get in touch or submit a pull request and we will add these configurations to the repository.

- `mamba`: Enables running all processes using mamba
- `singularity`: Enables running all processes using singularity
- `podman`: Enables running all processes using podman
- `shifter`: Enables running all processes using shifter
- `charliecloud`: Enables running all processes using charliecloud

Caution: Notice the number of hyphens used between `--input` and `-profile`. `--input` is an *Autometa* workflow parameter where as `-profile` is a *nextflow* argument. This difference in hyphens is true for passing in all arguments to the *Autometa* workflow and *nextflow*, respectively.

Running the pipeline

After you are finished double-checking your parameter settings, click “Launch” at the top right of web based GUI page, or “Launch workflow” at the bottom of the page. After returning to the terminal you should be provided the option `Do you want to run this command now? [y/n]` enter `y` to begin the pipeline.

This process will lead to nf-core tools creating a file named `nf-params.json`. This file contains your specified parameters that differed from the pipeline’s defaults. This file can also be manually modified and/or shared to allow reproducible configuration of settings (e.g. among members within a lab sharing the same server).

Additionally all Autometa specific pipeline parameters can be used as command line arguments using the `nextflow run ...` command by prepending the parameter name with two hyphens (e.g. `--outdir /path/to/output/workflow/results`)

Caution: If you are restarting from a previous run, **DO NOT FORGET** to also add the `-resume` flag to the nextflow run command. **Notice only 1 hyphen is used** with the `-resume` nextflow parameter!

Note: You can run the `KwanLab/Autometa` project without using nf-core if you already have a correctly formatted parameters file. (like the one generated from `nf-core launch ...`, i.e. `nf-params.json`)

```
nextflow run KwanLab/Autometa -params-file nf-params.json -profile slurm -resume
```

1.2.6 Advanced

Parallel computing and computer resource allotment

While you might want to provide Autometa all the compute resources available in order to get results faster, that may or may not actually achieve the fastest run time.

Within the Autometa pipeline, parallelization happens by providing all the assemblies at once to software that internally handles parallelization.

The Autometa pipeline will try and use all resources available to individual pipeline modules. Each module/process has been pre-assigned resource allotments via a low/medium/high tag. This means that even if you don’t select for the pipeline to run in parallel some modules (e.g. DIAMOND BLAST) may still use multiple cores.

- The maximum number of CPUs that any single module can use is defined with the `--max_cpus` option (default: 4).
- You can also set `--max_memory` (default: 16GB)
- `--max_time` (default: 240h). `--max_time` refers to the maximum time *each process* is allowed to run, *not* the execution time for the the entire pipeline.

Databases

Autometa uses the following NCBI databases throughout its pipeline:

- **Non-redundant nr database**
    - ftp.ncbi.nlm.nih.gov/blast/db/FASTA/nr.gz
- **prot.accession2taxid.gz**
    - ftp.ncbi.nih.gov/pub/taxonomy/accession2taxid/prot.accession2taxid.gz
- **nodes.dmp, names.dmp and merged.dmp - Found within**
    - ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

If you are running autometa for the first time you'll have to download these databases. You may use `autometa-update-databases --update-ncbi`. This will download the databases to the default path. You can check the default paths using `autometa-config --print`. If you need to change the default download directory you can use `autometa-config --section databases --option ncbi --value <path/to/new/ncbi_database_directory>`. See `autometa-update-databases -h` and `autometa-config -h` for full list of options.

In your `nf-params.json` file you also need to specify the directory where the different databases are present. Make sure that the directory path contains the following databases:

- Diamond formatted nr file => nr.dmnd
- Extracted files from tarball taxdump.tar.gz
- prot.accession2taxid.gz

```
{
    "single_db_dir" = "$HOME/Autometa/autometa/databases/ncbi"
}
```

Note: Find the above section of code in `nf-params.json` and update this path to the folder with all of the downloaded/formatted NCBI databases.

CPUs, Memory, Disk

Note: Like nf-core pipelines, we have set some automatic defaults for Autometa's processes. These are dynamic and each process will try a second attempt using more resources if the first fails due to resources. Resources are always capped by the parameters (show with defaults):

- `--max_cpus = 2`
- `--max_memory = 6.GB`
- `--max_time = 48.h`

The best practice to change the resources is to create a new config file and point to it at runtime by adding the flag `-c path/to/custom/file.config`

For example, to give all resource-intensive (i.e. having `label process_high`) jobs additional memory and cpus, create a file called `process_high_mem.config` and insert

```
process {
    withLabel:process_high {
        memory = 200.GB
        cpus = 32
    }
}
```

Then your command to run the pipeline (assuming you've already run `nf-core launch KwanLab/Autometa` which created a `nf-params.json` file) would look something like:

```
nextflow run KwanLab/Autometa -params-file nf-params.json -c process_high_mem.config
```

> **Caution:** If you are restarting from a previous run, **DO NOT FORGET** to also add the `-resume` flag to the nextflow run command.
>
> **Notice only 1 hyphen is used** with the `-resume` nextflow parameter!

For additional information and examples see Tuning workflow resources

Additional Autometa parameters

Up to date descriptions and default values of Autometa's nextflow parameters can be viewed using the following command:

```
nextflow run KwanLab/Autometa -r main --help
```

You can also adjust other pipeline parameters that ultimately control how binning is performed.

`params.length_cutoff` : Smallest contig you want binned (default is 3000bp)

`params.kmer_size` : kmer size to use

`params.norm_method` : Which kmer frequency normalization method to use. See *Advanced Usage* section for details

`params.pca_dimensions` : Number of dimensions of which to reduce the initial k-mer frequencies matrix (default is `50`). See *Advanced Usage* section for details

`params.embedding_method` : Choices are `sksne`, `bhsne`, `umap`, `densmap`, `trimap` (default is `bhsne`) See *Advanced Usage* section for details

`params.embedding_dimensions` : Final dimensions of the kmer frequencies matrix (default is `2`). See *Advanced Usage* section for details

`params.kingdom` : Bin contigs belonging to this kingdom. Choices are `bacteria` and `archaea` (default is `bacteria`).

`params.clustering_method` : Cluster contigs using which clustering method. Choices are "dbscan" and "hdbscan" (default is "dbscan"). See *Advanced Usage* section for details

`params.binning_starting_rank` : Which taxonomic rank to start the binning from. Choices are `superkingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species` (default is `superkingdom`). See *Advanced Usage* section for details

`params.classification_method` : Which clustering method to use for unclustered recruitment step. Choices are `decision_tree` and `random_forest` (default is `decision_tree`). See *Advanced Usage* section for details

`params.completeness` : Minimum completeness needed to keep a cluster (default is at least 20% complete, e.g. `20`). See *Advanced Usage* section for details `params.purity` : Minimum purity needed to keep a cluster (default is at least 95% pure, e.g. `95`). See *Advanced Usage* section for details

`params.cov_stddev_limit` : Which clusters to keep depending on the coverage std.dev (default is 25%, e.g. `25`). See *Advanced Usage* section for details

`params.gc_stddev_limit` : Which clusters to keep depending on the GC% std.dev (default is 5%, e.g. `5`). See *Advanced Usage* section for details

Customizing Autometa's Scripts

In case you want to tweak some of the scripts, run on your own scheduling system or modify the pipeline you can clone the repository and then run nextflow directly from the scripts as below:

```
# Clone the autometa repository into current directory
git clone git@github.com:KwanLab/Autometa.git

# Modify some code
# e.g. one of the local modules
code $HOME/Autometa/modules/local/align_reads.nf

# Generate nf-params.json file using nf-core
nf-core launch $HOME/Autometa

# Then run nextflow
nextflow run $HOME/Autometa -params-file nf-params.json -profile slurm
```

Note: If you only have a few metagenomes to process and you would like to customize Autometa's behavior, it may be easier to first try customization of the *Bash Workflow*

Useful options

`-c` : In case you have configured nextflow with your executor (see *Configuring your process executor*) and have made other modifications on how to run nextflow using your `nexflow.config` file, you can specify that file using the `-c` flag

To see all of the command line options available you can refer to nexflow CLI documentation

Resuming the workflow

One of the most powerful features of nextflow is resuming the workflow from the last completed process. If your pipeline was interrupted for some reason you can resume it from the last completed process using the resume flag (`-resume`). Eg, `nextflow run KwanLab/Autometa -params-file nf-params.json -c my_other_parameters.config -resume`

Execution Report

After running nextflow you can see the execution statistics of your autometa run, including the time taken, CPUs used, RAM used, etc separately for each process. Nextflow will generate summary, timeline and trace reports automatically for you in the `${params.outdir}/trace` directory. You can read more about this in the nextflow docs on execution reports.

Visualizing the Workflow

You can visualize the entire workflow ie. create the directed acyclic graph (DAG) of processes from the written DOT file. First install Graphviz (`mamba install -c anaconda graphviz`) then do `dot -Tpng < pipeline_info/autometa-dot > autometa-dag.png` to get the in the `png` format.

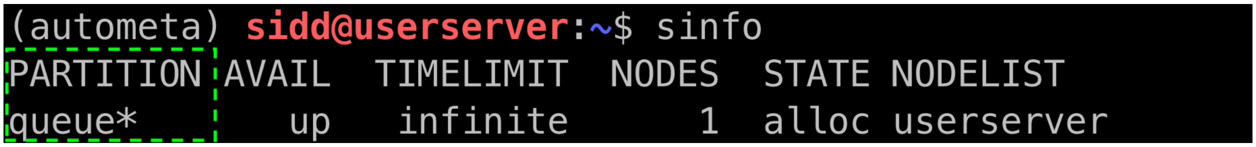
Configuring your process executor

For nextflow to run the Autometa pipeline through a job scheduler you will need to update the respective `profile` section in nextflow's config file. Each `profile` may be configured with any available scheduler as noted in the nextflow executors docs. By default nextflow will use your local computer as the 'executor'. The next section briefly walks through nextflow executor configuration to run with the slurm job scheduler.

We have prepared a template for `nextflow.config` which you can access from the KwanLab/Autometa GitHub repository using this nextflow.config template. Go ahead and copy this file to your desired location and open it in your favorite text editor (eg. Vim, nano, VSCode, etc).

SLURM

This allows you to run the pipeline using the SLURM resource manager. To do this you'll first needed to identify the slurm partition to use. You can find the available slurm partitions by running `sinfo`. Example: On running `sinfo` on our cluster we get the following:



```
(autometa) sidd@userserver:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
queue*       up   infinite      1  alloc userserver
```

The slurm partition available on our cluster is `queue`. You'll need to update this in `nextflow.config`.

```
profiles {
    // Find this section of code in nextflow.config
    slurm {
        process.executor       = "slurm"
        // NOTE: You can determine your slurm partition (e.g. process.queue) with the␣
↪`sinfo` command
        // Set SLURM partition with queue directive.
        process.queue = "queue" // <<-- change this to whatever your partition is called
        // queue is the slurm partition to use in our case
        docker.enabled         = true
        docker.userEmulation   = true
        singularity.enabled    = false
        podman.enabled         = false
        shifter.enabled        = false
```

(continues on next page)

(continued from previous page)

```
        charliecloud.enabled    = false
        executor {
            queueSize = 8
        }
    }
}
```

More parameters that are available for the slurm executor are listed in the nextflow executor docs for slurm.

Docker image selection

Especially when developing new features it may be necessary to run the pipeline with a custom docker image. Create a new image by navigating to the top Autometa directory and running `make image`. This will create a new Autometa Docker image, tagged with the name of the current Git branch.

To use this tagged version (or any other Autometa image tag) add the argument `--autometa_image tag_name` to the nextflow run command

1.3 Bash Workflow

1.3.1 Getting Started

1. *Compute Environment Setup*
2. *Download Workflow Template*
3. *Configure Required Inputs*

Compute Environment Setup

If you have not previously installed/used mamba, you can get it from Mambaforge.

You may either create a new mamba environment named "autometa"…

```
mamba create -n autometa -c conda-forge -c bioconda autometa
# Then, once mamba has finished creating the environment
# you may activate it:
mamba activate autometa
```

... or install Autometa into any of your existing environments.

This installs Autometa in your current active environment:

```
mamba install -c conda-forge -c bioconda autometa
```

The next command installs Autometa in the provided environment:

```
mamba install -n <your-env-name> -c conda-forge -c bioconda autometa
```

Download Workflow Template

To run Autometa using the bash workflow you will simply need to download and configure the workflow template to your metagenomes specifications.

- autometa.sh
- autometa-large-data-mode.sh

Here are a few download commands if you do not want to navigate to the workflow on GitHub

via curl

```
curl -o autometa.sh https://raw.githubusercontent.com/KwanLab/Autometa/main/workflows/
↪autometa.sh
```

via wget

```
wget https://raw.githubusercontent.com/KwanLab/Autometa/main/workflows/autometa.sh
```

Note: The `autometa-large-data-mode` workflow is also available and is configured similarly to the `autometa` bash workflow.

Configure Required Inputs

The Autometa bash workflow requires the following input file and directory paths. To see how to prepare each input, see *Data preparation*

1. Assembly (`assembly`)
2. Alignments (`bam`)
3. ORFs (`orfs`)
4. Diamond blastp results table (`blast`)
5. NCBI database directory (`ncbi`)
6. Input sample name (`simpleName`)
7. Output directory (`outdir`)

1.3.2 Data preparation

1. *Metagenome Assembly* (`assembly`)
2. *Alignments Preparation* (`bam`)
3. *ORFs* (`orfs`)
4. *Diamond blastp Preparation* (`blast`)
5. *NCBI Preparation* (`ncbi`)

Metagenome Assembly

You will first need to assemble your shotgun metagenome, to provide to Autometa as input.

The following is a typical workflow for metagenome assembly:

1. Trim adapter sequences from the reads

   We usually use Trimmomatic.

2. Quality check the trimmed reads to ensure the adapters have been removed

   We usually use FastQC.

3. Assemble the trimmed reads

   We usually use MetaSPAdes which is a part of the SPAdes package.

4. Check the quality of your assembly (Optional)

   We usually use metaQuast for this (use `--min-contig 1` option to get an accurate N50). This tool can compute a variety of assembly statistics one of which is N50. This can often be useful for selecting an appropriate length cutoff value for pre-processing the metagenome.

Alignments Preparation

Note: The following example requires `bwa`, `kart` and `samtools`

```
mamba install -c bioconda bwa kart samtools
```

```
# First index metagenome assembly
bwa index \
    -b 550000000000 \ # block size for the bwtsw algorithm (effective with -a bwtsw)␣
↪[default=10000000]
    metagenome.fna     # Path to input metagenome

# Now perform alignments (we are using kart, but you can use another alignment tool if␣
↪you'd like)
kart \
    -i metagenome.fna                    \ # Path to input metagenome
    -t 20                                \ # Number of cpus to use
    -f /path/to/forward_reads.fastq.gz  \ # Path to forward paired-end reads
    -f2 /path/to/reverse_reads.fastq.gz \ # Path to reverse paired-end reads
    -o alignments.sam                      # Path to alignments output

# Now sort alignments and convert to bam format
samtools sort \
    -@ 40              \ # Number of cpus to use
    -m 10G             \ # Amount of memory to use
    alignments.sam     \ # Input alignments file path
    -o alignments.bam     # Output alignments file path
```

ORFs

Note: The following example requires `prodigal`. e.g. `mamba install -c bioconda prodigal`

```
prodigal -i metagenome.fna \
    -f "gbk" \
    -d "metagenome.orfs.fna" \
    -o "metagenome.orfs.gbk" \
    -a "metagenome.orfs.faa" \ # This generated file is required as input to the bash␣
↪workflow
    -s "metagenome.all_orfs.txt"
```

Diamond blastp Preparation

Note: The following example requires `diamond`. e.g. `mamba install -c bioconda diamond`

```
diamond blastp \
    --query "metagenome.orfs.faa" \ # See prodigal output from above
    --db /path/to/nr.dmnd         \ # See NCBI section
    --threads <num cpus to use>   \
    --out blastp.tsv # This generated file is required as input to the bash workflow
```

NCBI Preparation

If you are running Autometa for the first time you'll have to download the NCBI databases.

```
# First configure where you want to download the NCBI databases
autometa-config \
    --section databases \
    --option ncbi \
    --value <path/to/your/ncbi/database/directory>

# Now download and format the NCBI databases
autometa-update-databases --update-ncbi
```

Note: You can check the default config paths using `autometa-config --print`.

See `autometa-update-databases -h` and `autometa-config -h` for full list of options.

The previous command will download the following NCBI databases:

- **Non-redundant nr database**
  - ftp.ncbi.nlm.nih.gov/blast/db/FASTA/nr.gz
- **prot.accession2taxid.gz**
  - ftp.ncbi.nih.gov/pub/taxonomy/accession2taxid/prot.accession2taxid.gz

- **nodes.dmp, names.dmp and merged.dmp - Found within**
    - ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

Input Sample Name

A crucial step prior to running the Autometa bash workflow is specifying the metagenome sample name and where to output Autometa's results.

```
# Default
simpleName="TemplateAssemblyName"
# Replace with your sample name
simpleName="MySample"
```

Note: The `simpleName` that is provided will be used as a prefix to all of the resulting autometa output files.

Output directory

Immediately following the `simpleName` parameter, you will need to specify where to write all results.

```
# Default
outdir="AutometaOutdir"
# Replace with your output directory...
outdir="MySampleAutometaResults"
```

1.3.3 Running the pipeline

After you are finished configuring/double-checking your parameter settings..

You may run the pipeline via bash:

```
bash autometa.sh
```

or submit the pipeline into a queue:

For example, with slurm:

```
sbatch autometa.sh
```

Caution: Make sure your mamba autometa environment is activated or the autometa entrypoints will not be available.

1.3.4 Additional parameters

You can also adjust other pipeline parameters that ultimately control how binning is performed. These are located at the top of the workflow just under the required inputs.

`length_cutoff` : Smallest contig you want binned (default is 3000bp)

`kmer_size` : kmer size to use

`norm_method` : Which kmer frequency normalization method to use. See *Advanced Usage* section for details

`pca_dimensions` : Number of dimensions of which to reduce the initial k-mer frequencies matrix (default is `50`). See *Advanced Usage* section for details

`embed_method` : Choices are `sksne`, `bhsne`, `umap`, `densmap`, `trimap` (default is `bhsne`) See *Advanced Usage* section for details

`embed_dimensions` : Final dimensions of the kmer frequencies matrix (default is `2`). See *Advanced Usage* section for details

`cluster_method` : Cluster contigs using which clustering method. Choices are "dbscan" and "hdbscan" (default is "dbscan"). See *Advanced Usage* section for details

`binning_starting_rank` : Which taxonomic rank to start the binning from. Choices are `superkingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species` (default is `superkingdom`). See *Advanced Usage* section for details

`classification_method` : Which clustering method to use for unclustered recruitment step. Choices are `decision_tree` and `random_forest` (default is `decision_tree`). See *Advanced Usage* section for details

`completeness` : Minimum completeness needed to keep a cluster (default is at least 20% complete, e.g. `20`). See *Advanced Usage* section for details

`purity` : Minimum purity needed to keep a cluster (default is at least 95% pure, e.g. `95`). See *Advanced Usage* section for details

`cov_stddev_limit` : Which clusters to keep depending on the coverage std.dev (default is 25%, e.g. `25`). See *Advanced Usage* section for details

`gc_stddev_limit` : Which clusters to keep depending on the GC% std.dev (default is 5%, e.g. `5`). See *Advanced Usage* section for details

Note: If you are configuring an autometa job using the `autometa-large-data-mode.sh` template, there will be an additional parameter called, `max_partition_size` (default=10,000). This is the maximum size partition the Autometa clustering algorithm will consider. Any taxon partitions larger than this setting will be skipped.

1.4 Bash Step by Step Tutorial

Here is the step by step tutorial of on running the entire pipeline manually through Bash. This is helpful in case you have your own files or just want to run a specific step.

If you would like to set up a run of the whole pipeline through Bash, see the *Bash Workflow* section.

Before running anything make sure you have activated the conda environment using `mamba activate autometa`.

See the Autometa Package Installation page for details on setting up your conda environment.

I will be going through this tutorial using the 78Mbp test dataset which can be found here https://drive.google.com/drive/u/2/folders/1McxKviIzkPyr8ovj8BG7n_IYk-QfHAgG. You only need to download `metagenome.fna.gz` from the above link and save it at a directory as per your liking. I'm saving it in `$HOME/tutorial/test_data/`. For instructions on how to download the dataset using command-line see the "Using command-line" section on *Benchmarking* page.

1.4.1 1. Length filter

The first step when running Autometa is the length filtering. This would remove any contigs that are below the length cutoff. This is useful in removing the noise from the data, as small contigs may have ambiguous kmer frequencies. The default cutoff if 3,000bp, ie. any contig that is smaller than 3,000bp would be removed.

Note: It is important that you alter the cutoff based on your N50. If your N50 is really small, e.g. 500bp (pretty common for soil assemblies), then you might want to lower your cutoff to somewhere near N50. The tradeoff with lowering the length cutoff, however, is a greater number of contigs which may make it more difficult for the dataset to be binned. As was shown in the Autometa paper, as assembly quality degrades so does the binning performance.

Use the following command to run the length-filter step:

```
autometa-length-filter \
    --assembly $HOME/tutorial/test_data/78mbp_metagenome.fna \
    --cutoff 3000 \
    --output-fasta $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --output-stats $HOME/tutorial/78mbp_metagenome.stats.tsv \
    --output-gc-content $HOME/tutorial/78mbp_metagenome.gc_content.tsv
```

Let us dissect the above command:

Flag	Input arguments	Requirement
`--assembly`	Path to metagenome assembly (nucleotide fasta) file	Required
`--cutoff`	Length cutoff for the filtered assembly. Default is 3,000bp	Optional
`--output-fasta`	Path to filtered metagenomic assembly that would be used for binning	Required
`--output-stats`	Path to assembly statistics table	Optional
`--output-gc-content`	Path to assembly contigs' GC content and length stats table	Optional

You can view the complete command-line options using `autometa-length-filter -h`

The above command generates the following files:

File	Description
78mbp_metagenome.filtered.fna	Length filtered metagenomic assembly to be used for binning
78mbp_metagenome.stats.tsv	Table describing the filtered metagenome assembly statistics
78mbp_metagenome.gc_content.tsv	Table of GC content and length of each contig in the filtered assembly

1.4.2 2. Coverage calculation

Coverage calculation for each contig is done to provide another parameter to use while clustering contigs.

from SPAdes

If you have used SPAdes to assemble your metagenome, you can use the following command to generate the coverage table:

```
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.fna \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --from-spades
```

from alignments.bed

If you have assembled your metagenome using some other assembler you can use one of the following commands to generate the coverage table.

```
# If you have already made a bed file
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --bed 78mbp_metagenome.bed \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --cpus 40
```

from alignments.bam

```
# If you have already made an alignment (bam file)
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --bam 78mbp_metagenome.bam \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --cpus 40
```

from alignments.sam

```
# If you have already made an alignment (sam file)
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --sam 78mbp_metagenome.sam \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --cpus 40
```

from paired-end reads

You may calculate coverage using forward and reverse reads with the assembled metagenome.

```
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --fwd-reads fwd_reads_1.fastq \
    --rev-reads rev_reads_1.fastq \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --cpus 40
```

In case you have multiple forward and reverse read pairs supply a comma-delimited list.

```
autometa-coverage \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --fwd-reads fwd_reads_1.fastq,fwd_reads_2.fastq \
    --rev-reads rev_reads_1.fastq,rev_reads_2.fastq \
    --out $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --cpus 40
```

Note:

1. No spaces should be used when providing the forward and reverse reads.
2. The lists of forward and reverse reads should be in the order corresponding to their respective reads pair.

Let us dissect the above commands:

Flag	Function
`--assembly`	Path to length filtered metagenome assembly
`--from-spades`	If the input assembly is generated using SPades then extract k-mer coverages from contig IDs
`--bed`	Path to alignments BED file
`--bed`	Path to alignments BAM file
`--sam`	Path to alignments SAM file
`--fwd-reads`	Path to forward reads
`--rev-reads`	Path to reverse reads
`--cpus`	Number of CPUs to use (default is to use all available CPUs)
`--out`	Path to coverage table of each contig

You can view the complete command-line options using `autometa-coverage -h`

The above command would generate the following files:

File	Description
78mbp_metagenome.coverages.tsv	Table with read or k-mer coverage of each contig in the metagenome

1.4.3 3. Generate Open Reading Frames (ORFs)

ORF calling using prodigal is performed here. The ORFs are needed for single copy marker gene detection and for taxonomic assignment.

Use the following command to run the ORF calling step:

```
autometa-orfs \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --output-nucls $HOME/tutorial/78mbp_metagenome.orfs.fna \
    --output-prots $HOME/tutorial/a78mbp_metagenome.orfs.faa \
    --cpus 40
```

Let us dissect the above command:

Flag	Function
`--assembly`	Path to length filtered metagenome assembly
`--output-nucls`	Path to nucleic acid sequence of ORFs
`--output-prots`	Path to amino acid sequence of ORFs
`--cpus`	Number of CPUs to use (default is to use all available CPUs)

You can view the complete command-line options using `autometa-orfs -h`

The above command would generate the following files:

File	Description
78mbp_metagenome.orfs.fna	Nucleic acid fasta file of ORFs
78mbp_metagenome.orfs.faa	Amino acid fasta file of ORFs

1.4.4 4. Single copy markers

Autometa uses single-copy markers to guide clustering, and does not assume that recoverable genomes will necessarily be “complete”. You first need to download the single-copy markers.

```
# Create a markers directory to hold the marker genes
mkdir -p $HOME/Autometa/autometa/databases/markers

# Change the default download path to the directory created above
autometa-config \
    --section databases \
    --option markers \
    --value $HOME/Autometa/autometa/databases/markers

# Download single-copy marker genes
autometa-update-databases --update-markers

# hmmpress the marker genes
hmmpress -f $HOME/Autometa/autometa/databases/markers/bacteria.single_copy.hmm
hmmpress -f $HOME/Autometa/autometa/databases/markers/archaea.single_copy.hmm
```

Use the following command to annotate contigs containing single-copy marker genes:

```
autometa-markers \
    --orfs $HOME/tutorial/78mbp_metagenome.orfs.faa \
    --kingdom bacteria \
    --hmmscan $HOME/tutorial/78mbp_metagenome.hmmscan.tsv \
    --out $HOME/tutorial/78mbp_metagenome.markers.tsv \
    --parallel \
    --cpus 4 \
    --seed 42
```

Let us dissect the above command:

Flag	Function	Requirement
`--orfs`	Path to fasta file containing amino acid sequences of ORFS	Required
`--kingdom`	Kingdom to search for markers. Choices bacteria (default) and archaea	Optional
`--hmmscan`	Path to hmmscan output table containing the respective kingdom single-copy marker annotations	Required
`--out`	Path to write filtered annotated markers corresponding to kingdom	Required
`--parallel`	Use hmmscan parallel option (default: False)	Optional
`--cpus`	Number of CPUs to use (default is to use all available CPUs)	Optional
`--seed`	Seed to set random state for hmmscan. (default: 42)	Optional

You can view the complete command-line options using `autometa-markers -h`

The above command would generate the following files:

File	Description
78mbp_metagenome.hmmscan.t	hmmscan output table containing the respective kingdom single-copy marker annotations
78mbp_metagenome.markers.tsv	Annotated marker table corresponding to the particular kingdom

1.4.5 5. Taxonomy assignment

5.1 BLASTP

Autometa assigns a taxonomic rank to each contig and then takes only the contig belonging to the specified kingdom (either bacteria or archaea) for binning. We found that in host-associated metagenomes, this step vastly improves the binning performance of Autometa (and other pipelines) because less eukaryotic or viral contigs will be placed into bacterial bins.

The first step for contig taxonomy assignment is a local alignment search of the ORFs against a reference database. This can be accelerated using diamond.

Create a diamond formatted database of the NCBI non-redundant (nr.gz) protein database.

```
diamond makedb \
    --in $HOME/Autometa/autometa/databases/ncbi/nr.gz \
    --db $HOME/Autometa/autometa/databases/ncbi/nr \
    --threads 40
```

Breaking down the above command:

Flag	Function
–in	Path to nr database
–db	Path to diamond formated nr database
-p	Number of processors to use

Note: `diamond makedb` will append `.dmnd` to the provided path of `--db`.

i.e. `--db /path/to/nr` will become `/path/to/nr.dmnd`

Run diamond blastp using the following command:

```
diamond blastp \
    --query $HOME/tutorial/78mbp_metagenome.orfs.faa \
    --db $HOME/Autometa/autometa/databases/ncbi/nr.dmnd \
    --evalue 1e-5 \
    --max-target-seqs 200 \
    --threads 40 \
    --outfmt 6 \
    --out $HOME/tutorial/78mbp_metagenome.blastp.tsv
```

Breaking down the above command:

Flag	Function
–query	Path to query sequence. Here, amino acid sequence of ORFs
–db	Path to diamond formatted nr database
–evalue	Maximum expected value to report an alignment
–max-target-seqs	Maximum number of target sequences per query to report alignments for
–threads	Number of processors to use
–outfmt	Output format of BLASTP results
–out	Path to BLASTP results

To see the complete list of acceptable output formats see Diamond GitHub Wiki. A complete list of all command-line options for Diamond can be found on its GitHub Wiki.

Caution: Autometa only parses output format 6 provided above as: `--outfmt 6`

The above command would generate the blastP table (`78mbp_metagenome.blastp.tsv`) in output format 6

5.2 Lowest Common Ancestor (LCA)

The second step in taxon assignment is determining each ORF's lowest common ancestor (LCA). This step uses the blastp results generated in the previous step to generate a table having the LCA of each ORF. As a default only the blastp hits (subject accessions) which are within 10% of the top bitscore are used. These subject accessions are translated to their respective taxids (`prot.accession2taxid.gz`) to be looked up in NCBI's taxonomy database (`nodes.dmp`). Each ORFs' list of taxids are then reduced to its lowest common ancestor via a range minimum query.

Note: For more details on the range minimum query algorithm, see the closed issue (#170) on Github and a walk-through on topcoder

---

Use the following command to get the LCA of each ORF:

```
autometa-taxonomy-lca \
    --blast $HOME/tutorial/78mbp_metagenome.blastp.tsv \
    --dbdir $HOME/Autometa/autometa/databases/ncbi/ \
    --lca-output $HOME/tutorial/78mbp_metagenome.lca.tsv \
    --sseqid2taxid-output $HOME/tutorial/78mbp_metagenome.lca.sseqid2taxid.tsv \
    --lca-error-taxids $HOME/tutorial/78mbp_metagenome.lca.errorTaxids.tsv
```

Let us dissect the above command:

Parameter	Function	Required (Y/N)
`--blast`	Path to diamond blastp output	Y
`--dbdir`	Path to NCBI databases directory	Y
`--lca-output`	Path to write lca output	Y
`--sseqid2taxid-outp`	Path to write qseqids sseqids to taxids translations table	N
`--lca-error-taxids`	Path to write table of blast table qseqids that were assigned root due to a missing taxid	N

You can view the complete command-line options using `autometa-taxonomy-lca -h`

The above command would generate a table (`78mbp_metagenome.lca.tsv`) having the name, rank and taxid of the LCA for each ORF.

5.3 Majority vote

The next step in taxon assignment is doing a modified majority vote to decide the taxonomy of each contig. This was developed to help minimize the effect of horizontal gene transfer (HGT). Briefly, the voting system helps assign the correct taxonomy to the contig from its component ORF classification. Even with highly divergent ORFs this allows for accurate kingdom level classification, enabling us to remove any eukaryotic contaminants or host DNA.

You can run the majority vote step using the following command:

```
autometa-taxonomy-majority-vote \
    --lca $HOME/tutorial/78mbp_metagenome.lca.tsv \
    --output $HOME/tutorial/78mbp_metagenome.votes.tsv \
    --dbdir $HOME/Autometa/autometa/databases/ncbi/
```

Let us dissect the above command:

Flag	Function
–lca	Path to LCA table
–output	Path to write majority vote table
–dbdir	Path to ncbi database directory

You can view the complete command-line options using `autometa-taxonomy-majority-vote -h`

The above command would generate a table (`78mbp_metagenome.votes.tsv`) having the taxid of each contig identified as per majority vote.

5.4 Split kingdoms

In this final step of taxon assignment we use the voted taxid of each contig to split the contigs into different kingdoms and write them as per the provided canonical rank.

```
autometa-taxonomy \
    --votes $HOME/tutorial/78mbp_metagenome.votes.tsv \
    --output $HOME/tutorial/ \
    --assembly $HOME/tutorial/78mbp_metagenome.filtered.fna \
    --prefix 78mbp_metagenome \
    --split-rank-and-write superkingdom \
    --ncbi $HOME/Autometa/autometa/databases/ncbi/
```

Let us dissect the above command:

Flag	Function	Requirement
`--votes`	Path to voted taxids table	Required
`--output`	Directory to output fasta files of split canonical ranks and taxonomy.tsv	Required
`--assembly`	Path to filtered metagenome assembly	Required
`--prefix`	prefix to use for each file written	Optional
`--split-rank-and-writ`	Split contigs by provided canonical-rank column then write to output directory	Optional
`--ncbi`	Path to ncbi database directory	Optional

Other options available for `--split-rank-and-write` are `phylum`, `class`, `order`, `family`, `genus` and `species`

If `--split-rank-and-write` is specified then it will split contigs by provided canonical-rank column then write a file corresponding that rank. Eg. Bacteria.fasta, Archaea.fasta, etc for `superkingdom`.

You can view the complete command-line options using `autometa-taxonomy -h`

File	Description
78mbp_metagenome.taxonomy.tsv	Table with taxonomic classification of each contig
78mbp_metagenome.bacteria.fna	Fasta file having the nucleic acid sequence of all bacterial contigs
78mbp_metagenome.unclassified.fı	Fasta file having the nucleic acid sequence of all contigs unclassified at kingdom level

In my case there are no non-bacterial contigs. For other datasets, `autometa-taxonomy` may produce other fasta files, for example Eukaryota.fasta and Viruses.fasta.

1.4.6 6. K-mer counting

A k-mer (ref) is just a sequence of k characters in a string (or nucleotides in a DNA sequence). It is known that contigs that belong to the same genome have similar k-mer composition (ref1 and ref2) . Here, we compute k-mer frequencies of only the bacterial contigs.

This step does the following:

1. Create a k-mer count matrix of $k^4/2$ dimensions using the specified k-mer length
2. Normalization of the k-mer count matrix to a normalized k-mer frequency matrix
3. Reduce the dimensions of k-mer frequencies using principal component analysis (PCA).
4. Embed the PCA dimensions into two dimensions to allow the ease of visualization and manual binning of the contigs (see ViZBin paper).

Use the following command to run the k-mer counting step:

```
autometa-kmers \
    --fasta $HOME/tutorial/78mbp_metagenome.bacteria.fna \
    --kmers $HOME/tutorial/78mbp_metagenome.bacteria.kmers.tsv \
    --size 5 \
    --norm-method am_clr \
    --norm-output $HOME/tutorial/78mbp_metagenome.bacteria.kmers.normalized.tsv \
    --pca-dimensions 50 \
    --embedding-method bhsne \
    --embedding-output $HOME/tutorial/78mbp_metagenome.bacteria.kmers.embedded.tsv \
    --cpus 40 \
    --seed 42
```

Let us dissect the above command:

Flag	Input arguments	Requirement
`--fasta`	Path to length filtered metagenome assembly	Required
`--kmers`	Path to k-mer frequency table	Required
`--size`	k-mer size in bp (default 5bp)	Optional
`--norm-output`	Path to normalized k-mer table	Required
`--norm-method`	Normalization method to transform kmer counts prior to PCA and embedding (default am_clr). Choices : ilr, clr and am_clr	Optional
`--pca-dimensi`	Number of dimensions to reduce to PCA feature space after normalization and prior to embedding (default: 50)	Optional
`--embedding-o`	Path to embedded k-mer table	Required
`--embedding-m`	Embedding method to reduce the k-mer frequencies. Choices: sksne, bhsne (default), umap, densmap and trimap.	Optional
`--cpus`	Number of CPUs to use (default is to use all available CPUs)	Optional
`--seed`	Set random seed for dimension reduction determinism (default 42). Useful in replicating the results	Optional

You can view the complete command-line options using `autometa-kmers -h`

The above command generates the following files:

File	Description
78mbp_metagenome.kmers.tsv	Table with raw k-mer counts of each contig
78mbp_metagenome.kmers.normalized.tsv	Table with normalized k-mer frequencies of each contig
78mbp_metagenome.kmers.embedded.tsv	Table with embedded k-mer frequencies of each contig

Advanced Usage

In the command used above k-mer normalization is being done using Autometa's implementation of the center log-ratio transform (am_clr). Other available normalization methods are isometric log-ratio transform (ilr, scikit-bio implementation) and center log-ratio transform (clr, scikit-bio implementation). Normalization method can be altered using the `--norm-method` flag.

In the above command k-mer embedding is being done using Barnes-Hut t-distributed Stochastic Neighbor Embedding (BH-tSNE). Other embedding methods that are available are Uniform Manifold Approximation and Projection (UMAP), densMAP (a density-preserving tool based on UMAP) and TriMap, a method that uses triplet constraints to form a low-dimensional embedding of a set of points. Two implementations of BH-tSNE are available, `bhsne` and `sksne` corresponding to the tsne and scikit-learn libraries, respectively. Embedding method can be altered using the `--embedding-method` flag.

Autometa uses a k-mer size of 5 and then embeds the resulting k-mer frequency table into 50 PCA dimensions which are then reduced to two dimentions. k-mer size can be altered using the `--size` flag, number of dimensions to reduce to PCA feature space after normalization and prior to embedding can be altered using the `--pca-dimensions` flag and the number of dimensions of which to reduce k-mer frequencies can be altered using the `--embedding-dimensions` flag.

Note: 1. Even though `bhsne` and `sksne` are the same embedding method (but different implementations) they appear to give very different results. We recommend using the former.

2. Providing a `0` to `--pca-dimensions` will skip the PCA step.

1.4.7 7. Binning

This is the step where contigs are binned into genomes via clustering. Autometa assesses genome bins by examining their completeness, purity, GC content std.dev. and coverage std.dev. A taxonomy table may also be used to selectively iterate through contigs based on their profiled taxon.

This step does the following:

1. Optionally iterate through contigs based on taxonomy
2. Bin contigs based on embedded k-mer coordinates and coverage
3. **Accept genome bins that pass the following metrics:**
    1. Above completeness threshold (`default=20.0`)
    2. Above purity threshold (`default=95.0`)
    3. Below GC content standard deviation threshold (`default=5.0`)
    4. Below coverage standard deviation threshold (`default=25.0`)
4. Unbinned contigs will be re-binned until no more acceptable genome bins are yielded

If you include a taxonomy table Autometa will attempt to further partition the data based on ascending taxonomic specificity (i.e. in the order superkingdom, phylum, class, order, family, genus, species) when binning unclustered contigs from a previous attempt. We found that this is mainly useful if you have a highly complex metagenome (lots of species), or you have several related species at similar coverage level.

Use the following command to perform binning:

```
autometa-binning \
    --kmers $HOME/tutorial/78mbp_metagenome.bacteria.kmers.embedded.tsv \
    --coverages $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --gc-content $HOME/tutorial/78mbp_metagenome.gc_content.tsv \
    --markers $HOME/tutorial/78mbp_metagenome.markers.tsv \
    --output-binning $HOME/tutorial/78mbp_metagenome.binning.tsv \
    --output-main $HOME/tutorial/78mbp_metagenome.main.tsv \
    --clustering-method dbscan \
    --completeness 20 \
    --purity 90 \
    --cov-stddev-limit 25 \
    --gc-stddev-limit 5 \
    --taxonomy $HOME/tutorial/78mbp_metagenome.taxonomy.tsv \
    --starting-rank superkingdom \
    --rank-filter superkingdom \
    --rank-name-filter bacteria
```

Let us dissect the above command:

Flag	Function	Requirement
`--kmers`	Path to embedded k-mer frequencies table	Required
`--coverages`	Path to metagenome coverages table	Required
`--gc-content`	Path to metagenome GC contents table	Required
`--markers`	Path to Autometa annotated markers table	Required
`--output-binnin`	Path to write Autometa binning results	Required
`--output-main`	Path to write Autometa main table	Required
`--clustering-me`	Clustering algorithm to use for recursive binning. Choices dbscan (default) and hdbscan	Optional
`--completeness`	completeness cutoff to retain cluster (default 20)	Optional
`--purity`	purity cutoff to retain cluster (default 95)	Optional
`--cov-stddev-li`	coverage standard deviation limit to retain cluster (default 25)	Optional
`--gc-stddev-lin`	GC content standard deviation limit to retain cluster (default 5)	Optional
`--taxonomy`	Path to Autometa assigned taxonomies table	Required
`--starting-rank`	Canonical rank at which to begin subsetting taxonomy (default: superkingdom)	Optional
`--rank-filter`	Canonical rank to subset by the value provided by `--rank-name-filter` default: superkingdom	Optional
`--rank-name-fil`	Only retrieve contigs with this value in the canonical rank column provided in `rank-filter` (default: bacteria)	Optional

You can view the complete command-line options using `autometa-binning -h`

The above command generates the following files:

1. `78mbp_metagenome.binning.tsv` contains the final binning results along with a few more metrics regarding each genome bin.

2. `78mbp_metagenome.main.tsv` which contains the feature table that was utilized during the genome binning process as well as the corresponding output predictions.

The following table describes each column for the resulting binning outputs. We'll start with the columns present in `78mbp_metagenome.binning.tsv` then describe the additional columns that are present in `78mbp_metagenome.main.tsv`.

Column	Description
Contig	Name of the contig in the input fasta file
Cluster	Genome bin assigned by autometa to the contig
Completeness	Estimated completeness of the Genome bin, based on single-copy marker genes
Purity	Estimated purity of the Genome bin, based on the number of single-copy marker genes that are duplicated in the cluster
coverage_stddev	Coverage standard deviation of the Genome bin
gc_content_stdd	GC content standard deviation of the Genome bin

In addition to the above columns `78mbp_metagenome.main.tsv` file has the following additional columns:

Column	Description
Coverage	Estimated coverage of the contig
gc_content	Estimated GC content of the contig
length	Estimated length of the contig
species	Assigned taxonomic species for the contig
genus	Assigned taxonomic genus for the contig
family	Assigned taxonomic family for the contig
order	Assigned taxonomic order for the contig
class	Assigned taxonomic class for the contig
phylum	Assigned taxonomic phylum for the contig
superkingdom	Assigned taxonomic superkingdom for the contig
taxid	Assigned NCBI taxonomy ID number for the contig
x_1	The first coordinate after dimension reduction
x_2	The second coordinate after dimension reduction

You can attempt to improve your genome bins with an unclustered recruitment step which uses features from existing genome bins to recruit unbinned contigs. Alternatively you can use these initial genome bin predictions and continue to the *Examining Results* section.

Advanced Usage

```
Completeness = Number of single copy marker genes present just once / Total number of␣
↪single copy marker genes

Purity = Number of single copy marker genes present more than once / Total number of␣
↪single copy marker genes
```

These are default parameters that autometa uses to accept clusters are 20% complete, 95% pure, below 25% coverage standard deviation and below 5% GC content standard deviation. These parameters can be altered using the flags, `--completeness`, `--purity`, `--cov-stddev-limit` and `--gc-stddev-limit`.

There are two binning algorithms to choose from Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN). The default is DBSCAN.

It is important to note that if recursively binning with taxonomy, only contigs at the specific taxonomic rank are analyzed and once the binning algorithm has moved on to the next rank, these are not considered until they fall under another taxonomic rank under consideration. I.e. Iterate through phyla. Contig of one phylum is only considered for that phylum then not for the rest of the phyla. If it is still unbinned at the Class rank, then it will be considered only during its respective Class's iteration. The canonical rank from which to start binning can be changed using the `--starting-rank` flag. The default is `superkingdom`.

1.4.8 8. Unclustered recruitment (Optional)

An unclustered recruitment step which uses features from existing genome bins is used to classify the unbinned contigs to the genome bins that were produced in the previous step. This step is optional and the results should be verified before proceeding with these results.

Note: The machine learning step has been observed to bin contigs that do not necessarily belong to the predicted genome. Careful inspection of coverage and taxonomy should be done before proceeding with these results.

Use the following command to run the unclustered recruitment step:

```
autometa-unclustered-recruitment \
    --kmers $HOME/tutorial/78mbp_metagenome.bacteria.kmers.normalized.tsv \
    --coverage $HOME/tutorial/78mbp_metagenome.coverages.tsv \
    --binning $HOME/tutorial/78mbp_metagenome.binning.tsv \
    --markers $HOME/tutorial/78mbp_metagenome.markers.tsv \
    --taxonomy $HOME/tutorial/78mbp_metagenome.taxonomy.tsv \
    --output-binning $HOME/tutorial/78mbp_metagenome.recruitment.binning.tsv \
    --output-features $HOME/tutorial/78mbp_metagenome.recruitment.features.tsv \
    --output-main $HOME/tutorial/78mbp_metagenome.recruitment.main.tsv \
    --classifier decision_tree \
    --seed 42
```

Let us dissect the above command:

Flag	Function	Required (Y/N)
`--kmers`	Path to normalized k-mer frequencies table	Y
`--coverages`	Path to metagenome coverages table	Y
`--binning`	Path to autometa binning output	Y
`--markers`	Path to Autometa annotated markers table	Y
`--output-binnin`	Path to write Autometa unclustered recruitment table	Y
`--taxonomy`	Path to taxonomy table	N
`--output-featur`	Path to write Autometa main table used during/after unclustered recruitment	N
`--output-main`	Path to write Autometa main table used during/after unclustered recruitment	N
`--classifier`	classifier to use for recruitment of contigs. Choices decision_tree (default) and random_forest	N
`--seed`	Seed to use for RandomState when initializing classifiers (default: 42)	N

You can view the complete command-line options using `autometa-unclustered-recruitment -h`

The above command would generate `78mbp_metagenome.recruitment.binning.tsv` and `78mbp_metagenome.recruitment.main.tsv`.

`78mbp_metagenome.recruitment.binning.tsv` contains the final predictions of `autometa-unclustered-recruitment`. `78mbp_metagenome.recruitment.features.tsv` is the feature table utilized during/after the unclustered recruitment algorithm. This represents unbinned contigs with their respective annotations and output predictions of their recruitment into a genome bin. The taxonomic features have been encoded using "one-hot encoding" or a presence/absence matrix where each column is a canonical taxonomic rank and its respective value for each row represents its presence or absence. Presence and absence are denoted with 1 and 0, respectively. Hence "one-hot" encoding being an encoding of presence and absence of the respective annotation type. In our case taxonomic designation.

The `78mbp_metagenome.recruitment.binning.tsv` file contains the following columns:

Column	Description
contig	Name of the contig in the input fasta file
cluster	Genome bin assigned by autometa to the contig
recruited_cluster	Genome bin assigned by autometa to the contig after unclustered recruitment step

Advanced Usage

The clustering method for the unclustered recruitment step can be performed either using a decision tree classifier (default) or using a random forst algorithm. The choice of method can be selected using the `--classifier` flag.

1.5 Databases

If you are running Autometa for the first time you will need to download and format a few databases. You may do this manually or using a few Autometa helper scripts. If you would like to use Autometa's scripts for this, you will first need to install Autometa (See *Installation*).

The following sections use a pair of commands to configure autometa such that the database is updated according to its respective path.

1.5.1 Markers

```
# Point Autometa to where you would like your markers database directory
autometa-config \
    --section databases --option markers \
    --value <path/to/your/markers/database/directory>

# Update your markers database directory
autometa-update-databases --update-markers
```

Links to these markers files and their associated cutoff values are below:

- bacteria single-copy-markers - link
- bacteria single-copy-markers cutoffs - link
- archaea single-copy-markers - link
- archaea single-copy-markers cutoffs - link

1.5.2 NCBI

```
# First configure where you want to download the NCBI databases
autometa-config \
    --section databases --option ncbi \
    --value <path/to/your/ncbi/database/directory>

# Now download and format the NCBI databases
autometa-update-databases --update-ncbi
```

Note: You can check the config paths using `autometa-config --print`.

See `autometa-update-databases -h` and `autometa-config -h` for full list of options.

The previous command will download the following NCBI databases:

- **Non-redundant nr database**
  - ftp.ncbi.nlm.nih.gov/blast/db/FASTA/nr.gz
- **prot.accession2taxid.gz**
  - ftp.ncbi.nih.gov/pub/taxonomy/accession2taxid/prot.accession2taxid.gz
- **nodes.dmp, names.dmp, merged.dmp and delnodes.dmp - Found within**
  - ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

After these files are downloaded, the `taxdump.tar.gz` tarball's files are extracted and the non-redundant protein database (`nr.gz`) is formatted as a diamond database (i.e. `nr.dmnd`). This will significantly speed-up the `diamond blastp` searches.

1.5.3 Genome Taxonomy Database (GTDB)

If you would like to incorporate the benefits of using the Genome Taxonomy Database, you can either run the following script or manually download the respective databases.

```
# First configure where you want to download the GTDB databases
autometa-config \
    --section databases --option gtdb \
    --value <path/to/your/gtdb/database/directory>

# To use a specific GTDB release
autometa-config \
    --section gtdb --option release \
    --value latest
    # Or --value r207 or --value r202, etc.

# Download and format the configured GTDB databases release
autometa-update-databases --update-gtdb
```

Note: You can check the default config paths using `autometa-config --print`.

See `autometa-update-databases -h` and `autometa-config -h` for full list of options.

The previous command will download the following GTDB databases and format the *gtdb_proteins_aa_reps.tar.gz* to generate *gtdb.dmnd* to be used by Autometa:

- **Amino acid sequences of representative genome**
    - gtdb_proteins_aa_reps.tar.gz
- **gtdb-taxdump.tar.gz from shenwei356/gtdb-taxdump**
    - gtdb-taxdump.tar.gz

Once unzipped *gtdb-taxdump.tar.gz* will have the taxdump files of all the respective GTDB releases. Make sure that the release you use is in line with the *gtdb_proteins_aa_reps.tar.gz* release version. It's better to always use the latest version.

All the taxonomy files for a specific taxonomy database should be in a single directory. You can now copy the taxdump files of the desired release version in the sample directory as *gtdb.dmnd*

Alternatively if you have manually downloaded *gtdb_proteins_aa_reps.tar.gz* and *gtdb-taxdump.tar.gz* you can run the following command to format the *gtdb_proteins_aa_reps.tar.gz* to generate *gtdb.dmnd* and make it ready for Autometa.

```
autometa-setup-gtdb --reps-faa <path/to/gtdb_proteins_aa_reps.tar.gz> --dbdir <path/to/
↪output_directory> --cpus 20
```

Note: Again Make sure that the formatted *gtdb_proteins_aa_reps.tar.gz* database and gtdb taxdump files are in the same directory.

1.6 Examining Results

1.6.1 Automappa

An interactive interface for exploration and refinement of metagenomes Automappa is a tool built to interface with Autometa output to help you explore your binning results.

For details, see the Automappa page

Note: The performance of Automappa may slow down when trying to visualize highly complex communities.

1.6.2 Visualize bins

To run the following commands you'll need to install R, Rstudio and ggplot2 package in R.

You can now run the following R scripts (preferably in RStudio) to examine your results.

```
# Load packages
library("ggplot2")

# Read the main binning table
filepath="/Users/sidd/Research/simulated/78mbp_metagenome.main.tsv"
data = read.table(filepath, header=TRUE, sep='\t')
```

(continues on next page)

(continued from previous page)

```
# Fill empty cells as unclustered
data$cluster <- sub("^$", "Unclustered", data$cluster)

ggplot(data, aes(x=x_1, y=x_2, color=cluster, group=cluster)) +
    geom_point(size=(sqrt(data$length))/100, shape=20, alpha=0.5) +
    theme_classic() + xlab('BH-tSNE X') + ylab('BH-tSNE Y') +
    guides( color = guide_legend( title = 'Genome Bin' ))
```

In the above chart each point represents a contig. They are plotted on two axes using results from dimension-reduction of k-mer frequencies. Rough differences between K-mer frequencies are utilized to guide Autometa's density-based binning algorithm. Points are also scaled in size according to their respective contig's length and colored by their assigned genome bin. You can see that there are some bins which are well-separated from others, while other bins are closer together. The latter cases may be worth investigating manually as multiple Autometa bins close together could actually be different parts of the same genome.

In addition to using nucleotide composition, Autometa uses coverage and can also use taxonomy to distinguish contigs with similar composition. We can also visualize these differences with R.

```
ggplot(data, aes(x=x_1, y=x_2, color=phylum, group=phylum)) +
    geom_point(size=(sqrt(data$length))/100, shape=20, alpha=0.5) +
    theme_classic() + xlab('BH-tSNE X') + ylab('BH-tSNE Y') +
    guides( color = guide_legend( title = 'Phylum' ))
```

In the above plot, we have now colored the points by taxonomic phylum, and this reveals that several clusters that are close together in BH-tSNE space are in fact quite divergent from one another (like bottom left). This is probably the basis for Autometa's assignment of separate bins in these cases.

In some cases, the contigs in a bin may in fact look divergent. You may want to manually examine cases such as these, but they could well be real if, for example, some contigs have few protein coding genes, or the organism is highly divergent from known sequences (see our paper here for some examples).

In this particular dataset, the coverages of all genomes are fairly similar, as revealed in the next plot:

```
ggplot(data, aes(x=coverage, y=gc_content, color=cluster, group=cluster)) +
    geom_point(size=(sqrt(data$length))/100, shape=20, alpha=0.5) +
    theme_classic() + xlab('Coverage') + ylab('GC content') +
    guides( color = guide_legend( title = 'Genome Bin' ))
```

In the above plot, the points are colored by genome bin again, and you can see that in this case, coverage is not much of a distinguishing feature. In other datasets, you may see closely related genomes at different coverages, which will be separable by Autometa.

1.7 Benchmarking

Note: The most recent Autometa benchmarking results covering multiple modules and input parameters are hosted on our KwanLab/metaBenchmarks Github repository and provide a range of analyses covering multiple stages and parameter sets. These benchmarks are available with their own respective modules so that the community may easily assess how Autometa's novel (`taxon-profiling`, `clustering`, `binning`, `refinement`) algorithms perform compared to current state-of-the-art methods. Tools were selected for benchmarking based on their relevance to environmental, single-assembly, reference-free binning pipelines.

1.7.1 Benchmarking with the `autometa-benchmark` module

Autometa includes the `autometa-benchmark` entrypoint, a script to benchmark Autometa taxon-profiling, clustering and binning-classification prediction results using clustering and classification evaluation metrics. To select the appropriate benchmarking method, supply the `--benchmark` parameter with the respective choice. The three benchmarking methods are detailed below.

Note: If you'd like to follow along with the benchmarking commands, you may download the test datasets using:

```
autometa-download-dataset \
    --community-type simulated \
    --community-sizes 78Mbp \
    --file-names reference_assignments.tsv.gz binning.tsv.gz taxonomy.tsv.gz \
    --dir-path $HOME/Autometa/autometa/datasets/simulated
```

This will download three files:

- `reference_assignments`: tab-delimited file containing contigs with their reference genome assignments. `cols: [contig, reference_genome, taxid, organism_name, ftp_path, length]`
- `binning.tsv.gz`: tab-delimited file containing contigs with Autometa binning predictions, `cols: [contig, cluster]`
- `taxonomy.tsv.gz`: tab-delimited file containing contigs with Autometa taxon-profiling predictions `cols: [contig, kingdom, phylum, class, order, family, genus, species, taxid]`

Taxon-profiling

Example benchmarking with simulated communities

```
# Set community size (see above for selection/download of other community types)
community_size=78Mbp

# Inputs
## NOTE: predictions and reference were downloaded using autometa-download-dataset
predictions="$HOME/Autometa/autometa/datasets/simulated/${community_size}/taxonomy.tsv.gz
↪" # required columns -> contig, taxid
reference="$HOME/Autometa/autometa/datasets/simulated/${community_size}/reference_
↪assignments.tsv.gz"
```

(continues on next page)

(continued from previous page)

```
ncbi=$HOME/Autometa/autometa/databases/ncbi

# Outputs
output_wide="${community_size}.taxon_profiling_benchmarks.wide.tsv.gz" # file path
output_long="${community_size}.taxon_profiling_benchmarks.long.tsv.gz" # file path
reports="${community_size}_taxon_profiling_reports" # directory path

autometa-benchmark \
    --benchmark classification \
    --predictions $predictions \
    --reference $reference \
    --ncbi $ncbi \
    --output-wide $output_wide \
    --output-long $output_long \
    --output-classification-reports $reports
```

Note: Using `--benchmark=classification` requires the path to a directory containing files (nodes.dmp, names.dmp, merged.dmp) from NCBI's taxdump tarball. This should be supplied using the `--ncbi` parameter.

Clustering

Example benchmarking with simulated communities

```
# Set community size (see above for selection/download of other community types)
community_size=78Mbp

# Inputs
## NOTE: predictions and reference were downloaded using autometa-download-dataset
predictions="$HOME/Autometa/autometa/datasets/simulated/${community_size}/binning.tsv.gz
↪" # required columns -> contig, cluster
reference="$HOME/Autometa/autometa/datasets/simulated/${community_size}/reference_
↪assignments.tsv.gz"

# Outputs
output_wide="${community_size}.clustering_benchmarks.wide.tsv.gz"
output_long="${community_size}.clustering_benchmarks.long.tsv.gz"

autometa-benchmark \
    --benchmark clustering \
    --predictions $predictions \
    --reference $reference \
    --output-wide $output_wide \
    --output-long $output_long
```

Binning

Example benchmarking with simulated communities

```
# Set community size (see above for selection/download of other community types)
community_size=78Mbp

# Inputs
## NOTE: predictions and reference were downloaded using autometa-download-dataset
predictions="$HOME/Autometa/autometa/datasets/simulated/${community_size}/binning.tsv.gz
↪" # required columns -> contig, cluster
reference="$HOME/Autometa/autometa/datasets/simulated/${community_size}/reference_
↪assignments.tsv.gz"

# Outputs
output_wide="${community_size}.binning_benchmarks.wide.tsv.gz"
output_long="${community_size}.binning_benchmarks.long.tsv.gz"

autometa-benchmark \
    --benchmark binning-classification \
    --predictions $predictions \
    --reference $reference \
    --output-wide $output_wide \
    --output-long $output_long
```

1.7.2 Autometa Test Datasets

Descriptions

Simulated Communities

Table 1: Autometa Simulated Communities

Community	Num. Genomes	Num. Control Sequences
78.125Mbp	21	4,044
156.25Mbp	38	3,573
312.50Mbp	85	7,708
625Mbp	166	17,590
1250Mbp	319	41,507
2500Mbp	656	67,702
5000Mbp	1,288	140,529
10000Mbp	2,638	285,262

You can download all the Simulated communities using this link. Individual communities can be downloaded using the links in the above table.

For more information on simulated communities, check the README.md located in the `simulated_communities` directory.

Synthetic Communities

51 bacterial isolates were assembled into synthetic communities which we've titled `MIX51`.

The initial synthetic community was prepared using a mixture of fifty-one bacterial isolates. The synthetic community's DNA was extracted for sequencing, assembly and binning.

You can download the MIX51 community using this link.

Download

Using `autometa-download-dataset`

Autometa is packaged with a built-in module that allows any user to download any of the available test datasets. To use retrieve these datasets one simply needs to run the `autometa-download-dataset` command.

For example, to download the reference assignments for a simulated community as well as the most recent Autometa binning and taxon-profiling predictions for this community, provide the following parameters:

```
# choices for simulated: 78Mbp,156Mbp,312Mbp,625Mbp,1250Mbp,2500Mbp,5000Mbp,10000Mbp
autometa-download-dataset \
    --community-type simulated \
    --community-sizes 78Mbp \
    --file-names reference_assignments.tsv.gz binning.tsv.gz taxonomy.tsv.gz \
    --dir-path simulated
```

This will download `reference_assignments.tsv.gz`, `binning.tsv.gz`, `taxonomy.tsv.gz` to the `simulated/78Mbp` directory.

- `reference_assignments`: tab-delimited file containing contigs with their reference genome assignments. `cols: [contig, reference_genome, taxid, organism_name, ftp_path, length]`
- `binning.tsv.gz`: tab-delimited file containing contigs with Autometa binning predictions, `cols: [contig, cluster]`
- `taxonomy.tsv.gz`: tab-delimited file containing contigs with Autometa taxon-profiling predictions `cols: [contig, kingdom, phylum, class, order, family, genus, species, taxid]`

Using `gdrive`

You can download the individual assemblies of different datasests with the help of `gdown` using command line (This is what `autometa-download-dataset` is using behind the scenes). If you have installed `autometa` using `mamba` then `gdown` should already be installed. If not, you can install it using `mamba install -c conda-forge gdown` or `pip install gdown`.

Example for the 78Mbp simulated community

1. Navigate to the 78Mbp community dataset using the link mentioned above.
2. **Get the file ID by navigating to any of the files and right clicking, then selecting the `get link` option.** This will have a `copy link` button that you should use. The link for the metagenome assembly (ie. `metagenome.fna.gz`) should look like this : `https://drive.google.com/file/d/15CB8rmQaHTGy7gWtZedfBJkrwr51bb2y/view?usp=sharing`
3. The file ID is within the `/` forward slashes between `file/d/` and `/`, e.g:

```
# Pasted from copy link button:
https://drive.google.com/file/d/15CB8rmQaHTGy7gWtZedfBJkrwr51bb2y/view?usp=sharing
#                 begin file ID ^ ------------------------------^ end file ID
```

4. Copy the file ID
5. Now that we have the File ID, you can specify the ID or use the `drive.google.com` prefix. Both should work.

```
file_id="15CB8rmQaHTGy7gWtZedfBJkrwr51bb2y"
gdown --id ${file_id} -O metagenome.fna.gz
# or
gdown https://drive.google.com/uc?id=${file_id} -O metagenome.fna.gz
```

Note: Unfortunately, at the moment `gdown` doesn't support downloading entire directories from Google drive. There is an open Pull request on the `gdown` repository addressing this specific issue which we are keeping a close eye on and will update this documentation when it is merged.

1.7.3 Advanced

Data Handling

Aggregating benchmarking results

When dataset index is unique

```
import pandas as pd
import glob
df = pd.concat([
    pd.read_csv(fp, sep="\t", index_col="dataset")
    for fp in glob.glob("*.clustering_benchmarks.long.tsv.gz")
])
df.to_csv("benchmarks.tsv", sep='\t', index=True, header=True)
```

When dataset index is *not* unique

```python
import pandas as pd
import os
import glob
dfs = []
for fp in glob.glob("*.clustering_benchmarks.long.tsv.gz"):
    df = pd.read_csv(fp, sep="\t", index_col="dataset")
    df.index = df.index.map(lambda fpath: os.path.basename(fpath))
    dfs.append(df)
df = pd.concat(dfs)
df.to_csv("benchmarks.tsv", sep='\t', index=True, header=True)
```

Downloading multiple test datasets at once

To download all of the simulated communities reference binning/taxonomy assignments as well as the Autometa v2.0 binning/taxonomy predictions all at once, you can provide the multiple arguments to `--community-sizes`.

e.g. `--community-sizes 78Mbp 156Mbp 312Mbp 625Mbp 1250Mbp 2500Mbp 5000Mbp 10000Mbp`

An example of this is shown in the bash script below:

```bash
# choices: 78Mbp,156Mbp,312Mbp,625Mbp,1250Mbp,2500Mbp,5000Mbp,10000Mbp
community_sizes=(78Mbp 156Mbp 312Mbp 625Mbp 1250Mbp 2500Mbp 5000Mbp 10000Mbp)

autometa-download-dataset \
    --community-type simulated \
    --community-sizes ${community_sizes[@]} \
    --file-names reference_assignments.tsv.gz binning.tsv.gz taxonomy.tsv.gz \
    --dir-path simulated
```

Generating new simulated communities

Communities were simulated using ART, a sequencing read simulator, with a collection of 3000 bacteria randomly retrieved. Genomes were retrieved until the provided total length was reached.

e.g. `-l 1250` would translate to 1250Mbp as the sum of total lengths for all bacterial genomes retrieved.

```bash
# Work out coverage level for art_illumina
# C = [(LN)/G]/2
# C = coverage
# L = read length (total of paired reads)
# G = genome size in bp
# -p  : indicate a paired-end read simulation or to generate reads from both ends of␣
↪amplicons
# -ss : HS25 -> HiSeq 2500 (125bp, 150bp)
# -f  : fold of read coverage simulated or number of reads/read pairs generated for each␣
↪amplicon
# -m  : the mean size of DNA/RNA fragments for paired-end simulations
# -s  : the standard deviation of DNA/RNA fragment size for paired-end simulations.
# -l  : the length of reads to be simulated
$ coverage = ((250 * reads) / (length * 1000000))
```

(continues on next page)

(continued from previous page)

```
$ art_illumina -p -ss HS25 -l 125 -f $coverage -o simulated_reads -m 275 -s 90 -i asm_
↪path
```

1.8 Installation

Currently Autometa package installation is supported by mamba, and docker. For installation using mamba, download mamba from Mambaforge.

Attention: If you are only trying to run the Autometa workflow, you should start at *Getting Started* before proceeding.

1.8.1 Direct installation (Quickest)

1. Install mamba

   ```
   wget "https://github.com/conda-forge/miniforge/releases/latest/download/
   ↪Mambaforge-$(uname)-$(uname -m).sh"
   bash Mambaforge-$(uname)-$(uname -m).sh
   ```

   Follow the installation prompts and when you get to this:

   ```
   Do you wish the installer to initialize Mambaforge
   by running conda init? [yes|no]
   [no] >>> yes
   ```

   This will require restarting the terminal, or resetting the terminal with the source command

   ```
   # To resolve the comment:
   # ==> For changes to take effect, close and re-open your current shell. <==
   # type:
   source ~/.bashrc
   ```

   Note: If you already have conda installed, you can install mamba as a drop-in replacement.

   ```
   conda -n base -c conda-forge mamba -y
   ```

2. Create a new environment with `autometa` installed:

   ```
   mamba create -c conda-forge -c bioconda -n autometa autometa
   ```

   Note: You may add the `bioconda` and `conda-forge` channels to your mamba config to simplify the command.

   ```
   mamba config --append channels bioconda
   mamba config --append channels conda-forge
   ```

Now mamba will search the `bioconda` and `conda-forge` channels alongside the defaults channel.

```
mamba create -n autometa autometa
```

---

3. Activate `autometa` environment:

```
mamba activate autometa
```

1.8.2 Install from source (using make)

Download and install mamba. Now run the following commands:

```
# Navigate to the directory where you would like to clone Autometa
cd $HOME

# Clone the Autometa repository
git clone https://github.com/KwanLab/Autometa.git

# Navigate into the cloned repository
cd Autometa

# create autometa mamba environment
make create_environment

# activate autometa mamba environment
mamba activate autometa

# install autometa source code in autometa environment
make install
```

Note: You can see a list of all available make commands by running `make` without any other arguments.

1.8.3 Install from source (full commands)

Download and install mamba. Now run the following commands:

```
# Navigate to the directory where you would like to clone Autometa
cd $HOME

# Clone the Autometa repository
git clone https://github.com/KwanLab/Autometa.git

# Navigate into the cloned repository
cd Autometa

# Construct the autometa environment from autometa-env.yml
mamba env create --file=autometa-env.yml
```

(continues on next page)

(continued from previous page)

```
# Activate environment
mamba activate autometa

# Install the autometa code base from source
python -m pip install . --ignore-installed --no-deps -vv
```

1.8.4 Building the Docker image

You can build a docker image for your clone of the Autometa repository.

1. Install Docker
2. Run the following commands

```
# Navigate to the directory where you need to clone Autometa
cd $HOME

# Clone the Autometa repository
git clone https://github.com/KwanLab/Autometa.git

# Navigate into the cloned repository
cd Autometa

# This will tag the image as jasonkwan/autometa:<your current branch>
make image

# (or the full command from within the Autometa repo)
docker build . -t jasonkwan/autometa:`git branch --show-current`
```

1.8.5 Testing Autometa

You can also check the installation using autometa's built-in unit tests. This is not at all necessary and is primarily meant for development and debugging purposes. To run the tests, however, you'll first need to install the following packages and download the test dataset.

```
# Activate your autometa mamba environment
mamba activate autometa

# List all make options
make

# Install dependencies for test environment
make test_environment

# Download test_data.json for unit testing to tests/data/
make unit_test_data_download
```

You can now run different unit tests using the following commands:

```
# Run all unit tests
make unit_test

# Run unit tests marked with entrypoint
make unit_test_entrypoints

# Run unit tests marked with WIP
make unit_test_wip
```

Note: As a shortcut you can also create the test environment and run **all** the unit tests using `make unit_test` command.

For more information about the above commands see the Contributing Guidelines page. Additional unit tests are provided in the test directory. These are designed to aid in future development of autometa.

1.9 Autometa Python API

1.9.1 Running modules

Many of the Autometa modules may be run standalone.

Simply pass in the `-m` flag when calling a script to signify to python you are running the script as an Autometa *module*.

I.e. `python -m autometa.common.kmers -h`

Note: Autometa has many *entrypoints* available that are utilized by the *Nextflow Workflow* and *Bash Workflow* . If you have installed autometa, all of these entrypoints will be available to you.

If you would like to get a better understanding of each entrypoint, we recommend reading the *Bash Step by Step Tutorial* section.

1.9.2 Using Autometa's Python API

Autometa's classes and functions are available after installation. To access these, do the same as importing any other python library.

Examples

Samtools wrapper

To incorporate a call to `samtools sort` inside of your python code using the Autometa samtools wrapper.

```
from autometa.common.external import samtools

# To see samtools.sort parameters try the commented command below:
# samtools.sort?

```

(continues on next page)

(continued from previous page)

```
# Run samtools sort command in ipython interpreter
samtools.sort(sam="<path/to/alignment.sam>", out="<path/to/output/alignment.bam>",␣
↪cpus=4)
```

Metagenome Description

Here is an example to easily assess your metagenome's characteristics using Autometa's Metagenome class

```
from autometa.common.metagenome import Metagenome

# To see input parameters, instance attributes and methods
# Metagenome?

# Create a metagenome instance
mg = Metagenome(assembly="/path/to/metagenome.fasta")

# To see available methods (ignore any elements in the list with a double underscore)
dir(mg)

# Get pandas dataframe of metagenome details.
metagenome_df = mg.describe()

metagenome_df.to_csv("path/to/metagenome_description.tsv", sep='\t', index=True,␣
↪header=True)
```

k-mer frequency counting, normalization, embedding

To quickly perform a k-mer frequency counting, normalization and embedding pipeline…

```
from autometa.common import kmers

# Count kmers
counts = kmers.count(
    assembly="/path/to/metagenome.fasta",
    size=5
)

# Normalize kmers
norm_df = kmers.normalize(
    df=counts,
    method="ilr"
)

# Embed kmers
embed_df = kmers.embed(
    norm_df,
    pca_dimensions=50,
    embed_dimensions=3,
```

(continues on next page)

(continued from previous page)

```
    method="densmap"
)
```

1.10 Usage

1.10.1 binning

summary.py

```
usage: summary.py

Summarize Autometa results writing genome fastas and their respective
taxonomies/assembly metrics for respective metagenomes

optional arguments:
  -h, --help            show this help message and exit
  --binning-main filepath
                        Path to Autometa binning main table (output from
                        --binning-main argument) (default: None)
  --markers filepath    Path to annotated markers respective to domain
                        (bacteria or archaea) binned (default: None)
  --metagenome filepath
                        Path to metagenome assembly (default: None)
  --dbdir dirpath       Path to user taxonomy database directory (Required for
                        retrieving metabin taxonomies) (default: None)
  --dbtype {ncbi,gtdb}  Taxonomy database type to use (NOTE: must correspond
                        to the same database type used during contig taxon
                        assignment.) (default: ncbi)
  --binning-column str  Binning column to use for grouping metabins (default:
                        cluster)
  --output-stats filepath
                        Path to write metabins stats table (default: None)
  --output-taxonomy filepath
                        Path to write metabins taxonomies table (default:
                        None)
  --output-metabins dirpath
                        Path to output directory. (Directory must not exist.
                        This directory will be created.) (default: None)
```

large_data_mode.py

```
usage: large_data_mode.py

Autometa Large-data-mode binning by contig set selection using max-partition-
size

optional arguments:
  -h, --help            show this help message and exit
  --kmers filepath      Path to k-mer counts table (default: None)
  --coverages filepath  Path to metagenome coverages table (default: None)
  --gc-content filepath
                        Path to metagenome GC contents table (default: None)
  --markers filepath    Path to Autometa annotated markers table (default:
                        None)
  --taxonomy filepath   Path to Autometa assigned taxonomies table (default:
                        None)
  --output-binning filepath
                        Path to write Autometa binning results (default: None)
  --output-main filepath
                        Path to write Autometa main table used during/after
                        binning (default: None)
  --clustering-method {dbscan,hdbscan}
                        Clustering algorithm to use for recursive binning.
                        (default: dbscan)
  --completeness 0 < float <= 100
                        completeness cutoff to retain cluster. e.g. cluster
                        completeness >= `completeness` (default: 20.0)
  --purity 0 < float <= 100
                        purity cutoff to retain cluster. e.g. cluster purity
                        >= `purity` (default: 95.0)
  --cov-stddev-limit float
                        coverage standard deviation limit to retain cluster
                        e.g. cluster coverage standard deviation <= `cov-
                        stddev-limit` (default: 25.0)
  --gc-stddev-limit float
                        GC content standard deviation limit to retain cluster
                        e.g. cluster GC content standard deviation <= `gc-
                        content-stddev-limit` (default: 5.0)
  --norm-method {am_clr,ilr,clr}
                        kmer normalization method to use on kmer counts
                        (default: am_clr)
  --pca-dims int        PCA dimensions to reduce normalized kmer frequencies
                        prior to embedding (default: 50)
  --embed-method {bhsne,umap,sksne,trimap}
                        kmer embedding method to use on normalized kmer
                        frequencies (default: bhsne)
  --embed-dims int      Embedding dimensions to reduce normalized kmers table
                        after PCA. (default: 2)
  --max-partition-size int
                        Maximum number of contigs to consider for a recursive
                        binning batch. (default: 10000)
  --starting-rank {superkingdom,phylum,class,order,family,genus,species}
```

(continues on next page)

(continued from previous page)

```
                        Canonical rank at which to begin subsetting taxonomy
                        (default: superkingdom)
  --reverse-ranks       Reverse order at which to split taxonomy by canonical-
                        rank. When `--reverse-ranks` is given, contigs will be
                        split in order of species, genus, family, order,
                        class, phylum, superkingdom. (default: False)
  --cache dirpath       Directory to store itermediate checkpoint files during
                        binning (If this is provided and the job fails, the
                        script will attempt to begin from the checkpoints in
                        this cache directory). (default: None)
  --binning-checkpoints filepath
                        File path to store itermediate contig binning results
                        (The `--cache` argument is required for this feature).
                        If `--cache` is provided without this argument, a
                        binning checkpoints file will be created. (default:
                        None)
  --rank-filter {superkingdom,phylum,class,order,family,genus,species}
                        Taxonomy column canonical rank to subset by provided
                        value of `--rank-name-filter` (default: superkingdom)
  --rank-name-filter RANK_NAME_FILTER
                        Only retrieve contigs with this name corresponding to
                        `--rank-filter` column (default: bacteria)
  --verbose             log debug information (default: False)
  --cpus int            Number of cores to use by clustering method (default
                        will try to use as many as are available) (default:
                        -1)
```

unclustered_recruitment.py

```
usage: unclustered_recruitment.py

Recruit unclustered contigs given metagenome annotations and Autometa binning
results. Note: All tables must contain a 'contig' column to be used as the
unique table index

optional arguments:
  -h, --help            show this help message and exit
  --kmers KMERS         Path to normalized kmer frequencies table. (default:
                        None)
  --coverage COVERAGE   Path to coverage table. (default: None)
  --binning BINNING     Path to autometa binning output [will look for
                        col='cluster'] (default: None)
  --markers MARKERS     Path to domain-specific markers table. (default: None)
  --output-binning OUTPUT_BINNING
                        Path to output unclustered recruitment table.
                        (default: None)
  --output-main OUTPUT_MAIN
                        Path to write Autometa main table used during/after
                        unclustered recruitment. (default: None)
  --output-features OUTPUT_FEATURES
```

(continues on next page)

(continued from previous page)

```
                        Path to write Autometa features table used during
                        unclustered recruitment. (default: None)
  --taxonomy TAXONOMY   Path to taxonomy table. (default: None)
  --taxa-dimensions TAXA_DIMENSIONS
                        Num of dimensions to reduce taxonomy encodings
                        (default: None)
  --additional-features [ADDITIONAL_FEATURES ...]
                        Path to additional features with which to add to
                        classifier training data. (default: [])
  --confidence CONFIDENCE
                        Percent confidence to allow classification (confidence
                        = num. consistent predictions/num. classifications)
                        (default: 1.0)
  --num-classifications NUM_CLASSIFICATIONS
                        Num classifications for predicting/validating contig
                        cluster recruitment (default: 10)
  --classifier {decision_tree,random_forest}
                        classifier to use for recruitment of contigs (default:
                        decision_tree)
  --kmer-dimensions KMER_DIMENSIONS
                        Num of dimensions to reduce normalized k-mer
                        frequencies (default: 50)
  --seed SEED           Seed to use for RandomState when initializing
                        classifiers. (default: 42)
```

recursive_dbscan.py

```
usage: recursive_dbscan.py

Perform marker gene guided binning of metagenome contigs using annotations
(when available) of sequence composition, coverage and homology.

optional arguments:
  -h, --help            show this help message and exit
  --kmers filepath      Path to embedded k-mers table (default: None)
  --coverages filepath  Path to metagenome coverages table (default: None)
  --gc-content filepath
                        Path to metagenome GC contents table (default: None)
  --markers filepath    Path to Autometa annotated markers table (default:
                        None)
  --output-binning filepath
                        Path to write Autometa binning results (default: None)
  --output-main filepath
                        Path to write Autometa main table used during/after
                        binning (default: None)
  --clustering-method {dbscan,hdbscan}
                        Clustering algorithm to use for recursive binning.
                        (default: dbscan)
  --completeness 0 < float <= 100
                        completeness cutoff to retain cluster. e.g. cluster
```

(continues on next page)

(continued from previous page)

```
                          completeness >= `completeness` (default: 20.0)
  --purity 0 < float <= 100
                          purity cutoff to retain cluster. e.g. cluster purity
                          >= `purity` (default: 95.0)
  --cov-stddev-limit float
                          coverage standard deviation limit to retain cluster
                          e.g. cluster coverage standard deviation <= `cov-
                          stddev-limit` (default: 25.0)
  --gc-stddev-limit float
                          GC content standard deviation limit to retain cluster
                          e.g. cluster GC content standard deviation <= `gc-
                          content-stddev-limit` (default: 5.0)
  --taxonomy filepath     Path to Autometa assigned taxonomies table (default:
                          None)
  --starting-rank {superkingdom,phylum,class,order,family,genus,species}
                          Canonical rank at which to begin subsetting taxonomy
                          (default: superkingdom)
  --reverse-ranks         Reverse order at which to split taxonomy by canonical-
                          rank. When `--reverse-ranks` is given, contigs will be
                          split in order of species, genus, family, order,
                          class, phylum, superkingdom. (default: False)
  --rank-filter {superkingdom,phylum,class,order,family,genus,species}
                          Taxonomy column canonical rank to subset by provided
                          value of `--rank-name-filter` (default: superkingdom)
  --rank-name-filter RANK_NAME_FILTER
                          Only retrieve contigs with this name corresponding to
                          `--rank-filter` column (default: bacteria)
  --verbose               log debug information (default: False)
  --cpus int              Number of cores to use by clustering method (default
                          will try to use as many as are available) (default:
                          -1)
```

large_data_mode_loginfo.py

```
usage: large_data_mode_loginfo.py

Retrieve clustering time stats from autometa.binning.recursive_dbscan err log

optional arguments:
  -h, --help       show this help message and exit
  --log LOG        Path to binning log file (If using slurm, this is typically
                   stderr output path) (default: None)
  --outdir OUTDIR  Directory to write runtime information tables (default: .)
  --prefix PREFIX  Prefix to prepend to runtime information tables (Do not use
                   a directory path as a prefix) (default: None)
  --overwrite      Overwrite existing log info table if it already exists
                   (default: False)
```

utilities.py

```
```

1.10.2 taxonomy

database.py

```
```

vote.py

```
usage: vote.py

Filter metagenome by taxonomy.

optional arguments:
  -h, --help            show this help message and exit
  --votes filepath      Input path to voted taxids table. should contain (at
                        least) 'contig' and 'taxid' columns (default: None)
  --assembly filepath   Path to metagenome assembly (nucleotide fasta).
                        (default: None)
  --output dirpath      Output directory to write specified canonical ranks
                        fasta files and taxon-binning results table (default:
                        None)
  --prefix str          prefix to use for each file written e.g.
                        `prefix`.taxonomy.tsv. Note: Do not use a directory
                        prefix. (default: None)
  --split-rank-and-write {superkingdom,phylum,class,order,family,genus,species}
                        If specified, will split contigs by provided
                        canonical-rank column then write to `output` directory
                        (default: None)
  --dbdir dirpath       Path to taxonomy database directory. (default:
                        NCBI_DIR)
  --dbtype {ncbi,gtdb}  Taxonomy database to use (default: ncbi)
```

majority_vote.py

```
usage: majority_vote.py

Script to assign taxonomy via a modified majority voting algorithm.

optional arguments:
  -h, --help            show this help message and exit
  --lca LCA             Path to LCA results table. (default: None)
  --output OUTPUT       Path to write voted taxid results table. (default:
                        None)
  --dbdir DBDIR         Path to taxonomy database directory. (default:
```

(continues on next page)

(continued from previous page)

```
                        NCBI_DIR)
  --dbtype {ncbi,gtdb}  Taxonomy database to use (default: ncbi)
  --orfs ORFS           Path to ORFs fasta containing amino-acid sequences to
                        be annotated. (Only required for prodigal version <
                        2.6) (default: None)
  --verbose             Add verbosity to logging stream. (default: False)
```

lca.py

```
usage: lca.py

Script to determine Lowest Common Ancestor

optional arguments:
  -h, --help            show this help message and exit
  --blast filepath      Path to BLAST results table respective to `orfs`.
                        (Note: The table provided must be in outfmt=6)
                        (default: None)
  --dbdir dirpath       Path to taxonomy databases directory. (default:
                        NCBI_DIR)
  --dbtype {ncbi,gtdb}  Taxonomy database to use (default: ncbi)
  --lca-output filepath
                        Path to write LCA results. (default: None)
  --sseqid2taxid-output filepath
                        Path to write qseqids sseqids to taxids translations
                        table (default: None)
  --lca-error-taxids filepath
                        Path to write table of blast table qseqids that were
                        assigned root due to a missing taxid (default: None)
  --verbose             Add verbosity to logging stream. (default: False)
  --force               Force overwrite if results already exist. (default:
                        False)
  --cache dirpath       Path to cache pickled LCA database objects. (default:
                        None)
  --only-prepare-cache  Only prepare the LCA database objects and write to
                        provided --cache parameter (default: False)
  --force-cache-overwrite
                        Force overwrite if results already exist. (default:
                        False)
```

ncbi.py

gtdb.py

```
usage: gtdb.py

optional arguments:
  -h, --help            show this help message and exit
  --reps-faa REPS_FAA  Path to directory containing GTDB ref genome animo acid
                        data sequences. Can be tarballed.
  --dbdir DBDIR        Path to output GTDB database directory
  --cpus CPUS          Number of cpus to use for diamond-formatting GTDB
                        database
```

1.10.3 config

environ.py

databases.py

```
usage: databases.py

Main script to configure Autometa database dependencies.

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG       </path/to/input/database.config> (default:
                        DEFAULT_FPATH)
  --dryrun              Log configuration actions but do not perform them.
                        (default: False)
  --update-all          Update all out-of-date databases. (NOTE: Does not
                        update GTDB) (default: False)
  --update-markers      Update out-of-date markers databases. (default: False)
  --update-ncbi         Update out-of-date ncbi databases. (default: False)
  --update-gtdb         Download and format the user-configured GTDB release
                        databases (default: False)
  --check-dependencies  Check database dependencies are satisfied. (default:
                        False)
  --no-checksum         Do not perform remote checksum comparisons to validate
                        databases are up-to-date. (default: False)
  --nproc NPROC         num. cpus to use for DB formatting. (default: 2)
  --out OUT             </path/to/output/database.config> (default: None)
```

(continues on next page)

(continued from previous page)

```
By default, with no arguments, will download/format databases into default
databases directory.
```

utilities.py

```
usage: utilities.py

Update Autometa configuration using provided arguments

optional arguments:
  -h, --help            show this help message and exit

Logging:
  --print               Print configuration without updating

Updating:
  --section {environ,databases,ncbi,markers,gtdb}
                        config section to update
  --option OPTION       option in `--section` to update
  --value VALUE         Value to update `--option`
```

1.10.4 common

external

hmmsearch.py

```
usage: hmmsearch.py

Filters domtblout generated from hmmsearch using provided cutoffs

optional arguments:
  -h, --help            show this help message and exit
  --domtblout DOMTBLOUT
                        Path to domtblout generated from hmmsearch -domtblout
                        <domtblout> ... <hmmfile> <seqdb>
  --cutoffs CUTOFFS     Path to cutoffs corresponding to hmmfile used with
                        hmmsearch <hmmfile> <seqdb>
  --seqdb SEQDB         Path to orfs seqdb used as input to hmmsearch ...
                        <hmmfile> <seqdb>
  --out OUT             Path to write table of markers passing provided
                        cutoffs
```

bowtie.py

```
usage: bowtie.py

Align provided reads to metagenome `assembly` and write alignments to
`sam`.NOTE: At least one reads file is required.

positional arguments:
  assembly              </path/to/assembly.fasta>
  database              </path/to/alignment.database>. Will construct database
                        at provided path if not found.
  sam                   </path/to/alignment.sam>

optional arguments:
  -h, --help            show this help message and exit
  -1 [FWD_READS ...], --fwd-reads [FWD_READS ...]
                        </path/to/forward-reads.fastq>
  -2 [REV_READS ...], --rev-reads [REV_READS ...]
                        </path/to/reverse-reads.fastq>
  -U [SE_READS ...], --se-reads [SE_READS ...]
                        </path/to/single-end-reads.fastq>
  --cpus CPUS           Num processors to use.
```

samtools.py

```
usage: samtools.py

Takes a sam file, sorts it and returns the output to a bam file

positional arguments:
  sam          </path/to/alignment.sam>
  bam          </path/to/output/alignment.bam>

optional arguments:
  -h, --help   show this help message and exit
  --cpus CPUS  Number of processors to use
```

hmmscan.py

```
usage: hmmscan.py

Retrieves markers with provided input assembly

positional arguments:
  orfs            </path/to/assembly.orfs.faa>
  hmmdb           </path/to/hmmpressed/hmmdb>
  cutoffs         </path/to/hmm/cutoffs.tsv>
  hmmscan         </path/to/hmmscan.tblout>
  markers         </path/to/markers.tsv>
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help      show this help message and exit
  --force         force overwrite of out filepath
  --cpus CPUS     num cpus to use
  --parallel      enable hmmer multithreaded parallelization
  --gnu-parallel  enable GNU parallelization
```

diamond.py

```
```

prodigal.py

```
usage: prodigal.py

Calls ORFs with provided input assembly

optional arguments:
  -h, --help            show this help message and exit
  --assembly filepath   Path to metagenome assembly (default: None)
  --output-nucls filepath
                        Path to output nucleotide ORFs (default: None)
  --output-prots filepath
                        Path to output amino-acid ORFs (default: None)
  --cpus int            Number of processors to use. (If more than one this
                        will parallelize prodigal using GNU parallel)
                        (default: 1)
  --force               Overwrite existing output ORF filepaths (default:
                        False)
```

bedtools.py

```
usage: bedtools.py

Compute genome coverage from sorted BAM file

optional arguments:
  -h, --help         show this help message and exit
  --ibam filepath    Path to sorted alignment.bam
  --bed filepath     Path to write alignment.bed; tab-delimited
                     cols=[contig,length]
  --output filepath  Path to output coverage.tsv
  --force-bed        force overwrite `bed`
  --force-cov        force overwrite `--output`
```

kmers.py

```
usage: kmers.py

Count k-mer frequencies of given `fasta`

optional arguments:
  -h, --help            show this help message and exit
  --fasta filepath      Metagenomic assembly fasta file (default: None)
  --kmers filepath      K-mers frequency tab-delimited table (will skip if
                        file exists) (default: None)
  --size int            k-mer size in bp (default: 5)
  --norm-output filepath
                        Path to normalized kmers table (will skip if file
                        exists) (default: None)
  --norm-method {ilr,clr,am_clr}
                        Normalization method to transform kmer counts prior to
                        PCA and embedding. ilr: isometric log-ratio transform
                        (scikit-bio implementation). clr: center log-ratio
                        transform (scikit-bio implementation). am_clr: center
                        log-ratio transform (Autometa implementation).
                        (default: am_clr)
  --pca-dimensions int  Number of dimensions to reduce to PCA feature space
                        after normalization and prior to embedding (NOTE:
                        Setting to zero will skip PCA step) (default: 50)
  --embedding-output filepath
                        Path to write embedded kmers table (will skip if file
                        exists) (default: None)
  --embedding-method {sksne,bhsne,umap,densmap,trimap}
                        embedding method [sk,bh]sne are corresponding
                        implementations from scikit-learn and tsne,
                        respectively. (default: bhsne)
  --embedding-dimensions int
                        Number of dimensions of which to reduce k-mer
                        frequencies (default: 2)
  --force               Whether to overwrite existing annotations (default:
                        False)
  --cpus int            num. processors to use. (default: 2)
  --seed int            Seed to set random state for dimension reduction
                        determinism. (default: 42)
```

exceptions.py

```
```

coverage.py

```
usage: coverage.py

Construct contig coverage table given an input `assembly` and provided files.
Provided files may include one from the list below:
1. `fwd_reads` and/or `rev_reads` and/or `se_reads`
2. `sam` - alignment of `assembly` and `reads` in SAM format
3. `bam` - alignment of `assembly` and `reads` in BAM format
4. `bed` - alignment of `assembly` and `reads` in BED format

optional arguments:
  -h, --help            show this help message and exit
  -f ASSEMBLY, --assembly ASSEMBLY
                        </path/to/metagenome.fasta>
  -1 [FWD_READS ...], --fwd-reads [FWD_READS ...]
                        </path/to/forwards-reads.fastq>
  -2 [REV_READS ...], --rev-reads [REV_READS ...]
                        </path/to/reverse-reads.fastq>
  -U [SE_READS ...], --se-reads [SE_READS ...]
                        </path/to/single-end-reads.fastq>
  --sam SAM             </path/to/alignments.sam>
  --bam BAM             </path/to/alignments.bam>
  --bed BED             </path/to/alignments.bed>
  --cpus CPUS           Num processors to use. (default: 2)
  --from-spades         Extract k-mer coverages from contig IDs. (Input
                        assembly is output from SPAdes)
  --out OUT             Path to write a table of coverages
```

metagenome.py

```
usage: metagenome.py

This script handles filtering by length and can calculate various metagenome
statistics.

optional arguments:
  -h, --help            show this help message and exit
  --assembly filepath   Path to metagenome assembly (nucleotide fasta).
                        (default: None)
  --output-fasta filepath
                        Path to output length-filtered assembly fasta file.
                        (default: None)
  --output-stats filepath
                        Path to output assembly stats table. (default: None)
  --output-gc-content filepath
```

(continues on next page)

(continued from previous page)

```
                        Path to output assembly contigs' GC content and
                        length. (default: None)
  --cutoff int          Cutoff to apply to length filter (default: 3000)
  --force               Overwrite existing files (default: False)
  --verbose             Log more information to terminal. (default: False)
```

markers.py

```
usage: markers.py

Annotate ORFs with kingdom-specific marker information

optional arguments:
  -h, --help            show this help message and exit
  --orfs ORFS           Path to a fasta file containing amino acid sequences
                        of open reading frames (default: None)
  --kingdom {bacteria,archaea}
                        kingdom to search for markers (default: bacteria)
  --hmmscan HMMSCAN     Path to hmmscan output table containing the respective
                        `kingdom` single-copy marker annotations. (default:
                        None)
  --out OUT             Path to write filtered annotated markers corresponding
                        to `kingdom`. (default: None)
  --dbdir DBDIR         Path to directory containing the single-copy marker
                        HMM databases. (default: MARKERS_DIR)
  --hmmdb HMMDB         Path to single-copy marker HMM databases. (default:
                        None)
  --cutoffs CUTOFFS     Path to single-copy marker cutoff tsv. (default: None)
  --force               Whether to overwrite existing provided annotations.
                        (default: False)
  --parallel            Whether to use hmmscan parallel option. (default:
                        False)
  --gnu-parallel        Whether to run hmmscan using GNU parallel. (default:
                        False)
  --cpus CPUS           Number of cores to use for parallel execution.
                        (default: 8)
  --seed SEED           Seed to set random state for hmmscan. (default: 42)
```

utilities.py

1.11 How to contribute

Autometa is an open-source project developed on GitHub. If you would like to help develop Autometa or have ideas for new features please see our contributing guidelines

Some good first issues are available on the KwanLab Autometa GitHub repository good first issues

If you are wanting to help develop Autometa, you will need these additional dependencies:

1.11.1 Documentation

Autometa builds documentation using readthedocs. You have to install the following to be able to build the docs

```
# Activate your autometa mamba environment
mamba activate autometa
# Install dependencies
mamba install -n autometa -c conda-forge \
    sphinx sphinx_rtd_theme
# List all make options
make
# Build documentation for autometa.readthedocs.io
make docs
```

`make docs`

This command runs sphinx and generates autometa documentation for autometa.readthedocs.io.

1.11.2 Unit tests

You will have to install certain dependencies as well as test data to be able to run and develop unit tests.

```
# Activate your autometa mamba environment
mamba activate autometa
# List all make options
make
# Install dependencies for test environment
make test_environment
# Download test_data.json for unit testing to tests/data/
make unit_test_data_download
```

You can now run different unit tests using the following commands:

```
# Run all unit tests
make unit_test
# Run unit tests marked with entrypoint
```

(continues on next page)

(continued from previous page)

```
make unit_test_entrypoints
# Run unit tests marked with WIP
make unit_test_wip
```

`make test_environment`

This command installs all the dependencies that you need to successfully run the unit tests.

`make unit_test_data_download`

This command downloads the test_data.json object that you need to run the Unit tests. This is a necessary step when wanting to run unit tests as the test_data.json file will hold many of the variables necessary to conduct these tests.

`make unit_test_data_build`

This is used to create your own `test_data.json` object locally. This step is NOT required for running unit tests, you can directly download the `test_data.json` object using the previous command. This command is needed in case you are changing file formats or adding more objects into the test suite. To do this you first need to download all the files from here in `tests/data/` and then run `make unit_test_data_build`. This would generate a similar `test_data.json` object that you get by running the previous command.

The above command is used to manually build the test_data.json file for unit testing. I.e. it will run the script make_test_data.py which will aggregate all of the files in the tests/data folder that have been downloaded from here. This is the first or perhaps 0th step when it comes to running the tests without an already generated `test_data.json` object as it generates the test_data.json file that is parsed to retrieve all of the pre-generated variables used for intermediate stages of the pipeline. This is done to reduce the test time and computational workload when running through the test suite.

`make unit_test`

This command runs all unit tests under the tests directory. This includes all tests marked as WIP or as entrypoints. However this will skip tests marked with the following decorator:

```
@pytest.mark.skip
def test_some_function(...):
    ...
```

`make unit_test_entrypoints`

This command runs the tests marked as entrypoints. This is denoted in pytest with the decorator:

```
@pytest.mark.entrypoint
def test_some_function_that_is_an_entrypoint(...):
...
```

Entrypoints correspond to the entry point functions listed out by ‘console scripts’ in setup.py. These entry point functions are aliased to provide more intuitive commands for the end user. These are important and sometimes referred to as “happy” tests because if one of these fail for the end-user, they will probably be quite unhappy and likely distrust the functionality of the rest of the codebase.

`make unit_test_wip`

This command runs the tests marked as work-in-progress (WIP). This is denoted in pytest with the decorator:

```
@pytest.mark.wip
def test_some_function_that_is_wip(...):
...
```

1.12 Autometa modules

1.12.1 autometa package

Subpackages

autometa.binning package

Submodules

autometa.binning.large_data_mode module

Autometa large-data-mode binning by selection of taxon sets using provided upper bound and determined lower bound

`autometa.binning.large_data_mode.`**`checkpoint`**(*checkpoints_df: pandas.DataFrame*, *clustered: pandas.DataFrame*, *rank: str*, *rank_name_txt: str*, *completeness: float*, *purity: float*, *coverage_stddev: float*, *gc_content_stddev: float*, *cluster_method: str*, *norm_method: str*, *pca_dimensions: int*, *embed_dimensions: int*, *embed_method: str*, *min_contigs: int*, *max_partition_size: int*, *binning_checkpoints_fpath: str*) → pandas.DataFrame

`autometa.binning.large_data_mode.`**`cluster_by_taxon_partitioning`**(*main: pandas.DataFrame*, *counts: pandas.DataFrame*, *markers: pandas.DataFrame*, *norm_method: str = 'am_clr'*, *pca_dimensions: int = 50*, *embed_dimensions: int = 2*, *embed_method: str = 'umap'*, *max_partition_size: int = 10000*, *completeness: float = 20.0*, *purity: float = 95.0*, *coverage_stddev: float = 25.0*, *gc_content_stddev: float = 5.0*, *starting_rank: str = 'superkingdom'*, *method: str = 'dbscan'*, *reverse_ranks: bool = False*, *cache: Optional[str] = None*, *binning_checkpoints_fpath: Optional[str] = None*, *n_jobs: int = -1*, *verbose: bool = False*) → pandas.DataFrame

Perform clustering of contigs by provided *method* and use metrics to filter clusters that should be retained via *completeness* and *purity* thresholds.

Parameters

- **main** (*pd.DataFrame*) – index=contig, cols=['coverage', 'gc_content'] taxa cols should be present if *taxonomy* is True. i.e. [taxid,superkingdom,phylum,class,order,family,genus,species]
- **counts** (*pd.DataFrame*) – contig kmer counts -> index_col='contig', cols=['AAAAA', 'AAAAT', ...] NOTE: columns will correspond to the selected k-mer count size. e.g. 3-mers would be ['AAA','AAT', ...]
- **markers** (*pd.DataFrame*) – wide format, i.e. index=contig cols=[marker,marker,...]
- **completeness** (*float, optional*) – Description of parameter *completeness* (the default is 20.).
- **purity** (*float, optional*) – purity threshold to retain cluster (the default is 95.0). e.g. cluster purity >= purity_cutoff
- **coverage_stddev** (*float, optional*) – cluster coverage threshold to retain cluster (the default is 25.0).
- **gc_content_stddev** (*float, optional*) – cluster GC content threshold to retain cluster (the default is 5.0).
- **starting_rank** (*str, optional*) – Starting canonical rank at which to begin subsetting taxonomy (the default is superkingdom). Choices are superkingdom, phylum, class, order, family, genus, species.
- **method** (*str, optional*) – Clustering *method* (the default is 'dbscan'). choices = ['dbscan','hdbscan']
- **reverse_ranks** (*bool, optional*) – False - [superkingdom,phylum,class,order,family,genus,species] (Default) True - [species,genus,family,order,class,phylum,superkingdom]
- **cache** (*str, optional*) – Directory to cache intermediate results
- **binning_checkpoints_fpath** (*str, optional*) – File path to binning checkpoints (checkpoints are only created if the *cache* argument is provided)
- **verbose** (*bool, optional*) – log stats for each recursive_dbscan clustering iteration

Returns

main with ['cluster','completeness','purity'] columns added

Return type

pd.DataFrame

Raises

- *TableFormatError* – No marker information is available for contigs to be binned.
- **FileNotFoundError** – Provided *binning_checkpoints_fpath* does not exist
- *TableFormatError* – No marker information is availble for contigs to be binned.

`autometa.binning.large_data_mode.`**`get_checkpoint_info`**(*checkpoints_fpath: str*) → Dict[str, Union[pandas.DataFrame, str]]

Retrieve checkpoint information from generated binning_checkpoints.tsv

Parameters
checkpoints_fpath (*str*) – Generated binning_checkpoints.tsv within cache directory

Returns
binning_checkpoints, starting canonical rank, starting rank name within starting canonical rank keys="binning_checkpoints", "starting_rank", "starting_rank_name_txt" values=pd.DataFrame, str, str

Return type
Dict[str, str, str]

`autometa.binning.large_data_mode.`**`get_kmer_embedding`**(*counts: pandas.DataFrame*, *cache_fpath: str*, *norm_method: str*, *pca_dimensions: int*, *embed_dimensions: int*, *embed_method: str*) → pandas.DataFrame

Retrieve kmer embeddings for provided counts by first performing kmer normalization with *norm_method* then PCA down to *pca_dimensions* until the normalized kmer frequencies are embedded to *embed_dimensions* using *embed_method*.

Parameters

- **counts** (*pd.DataFrame*) – Kmer counts where index column is 'contig' and each column is a kmer count.
- **cache_fpath** (*str*) – Path to cache embedded kmers table for later look-up/inspection.
- **norm_method** (*str*) – normalization transformation to use on kmer counts. Choices include 'am_clr', 'ilr' and 'clr'. See :func:kmers.normalize for more details.
- **pca_dimensions** (*int*) – Number of dimensions by which to initially reduce normalized kmer frequencies (Must be greater than *embed_dimensions*).
- **embed_dimensions** (*int*) – Embedding dimensions by which to reduce normalized PCA-transformed kmer frequencies (Must be less than *pca_dimensions*).
- **embed_method** (*str*) – Embedding method to use on normalized, PCA-transformed kmer frequencies. Choices include 'bhsne', 'sksne' and 'umap'. See :func:kmers.embed for more details.

Returns
[description]

Return type
pd.DataFrame

`autometa.binning.large_data_mode.`**`main`**()

autometa.binning.large_data_mode_loginfo module

Autometa module: autometa-large-data-mode-binning-loginfo

Generates tabular metadata from logfile for large-data-mode task (e.g. slurm_job.stderr)

`autometa.binning.large_data_mode_loginfo.`**`add_clustering_runtime_summary_info`**(*clustering_df: pandas.DataFrame*, *totals: pandas.DataFrame*) → pandas.DataFrame

Retrieve information about the clustering that took the longest.

Parameters

- **`clustering_df`** (*pd.DataFrame*) – Clustering runtime info retrieved from logfile
- **`totals`** (*pd.DataFrame*) – runtime totals summary table

Returns

runtime totals summary table updated with clustering runtime info

Return type

pd.DataFrame

`autometa.binning.large_data_mode_loginfo.`**`add_embedding_runtime_summary_info`**`(`*embedding_df: pandas.DataFrame*, *totals: pandas.DataFrame*`)` → pandas.DataFrame

Retrieve information about the embeddings that took the longest.

Parameters

- **`embedding_df`** (*pd.DataFrame*) – Embedding info retrieved from logfile
- **`totals`** (*pd.DataFrame*) – runtime totals summary table

Returns

runtime totals summary table updated with embedding info

Return type

pd.DataFrame

`autometa.binning.large_data_mode_loginfo.`**`format_total_times`**`(`*total_times: list*, *max_partition_size: str*`)` → pandas.DataFrame

Format total runtimes from timedelta objects to hours

Parameters

- **`total_times`** (*list*) – Runtime totals per algorithm during large-data-mode
- **`max_partition_size`** (*str*) – Partition size parameter retrieved from log file

Returns

Formatted dataframe of timedelta objects and hours per algorithm in large-data-mode run.

Return type

pd.DataFrame

`autometa.binning.large_data_mode_loginfo.`**`get_loginfo`**`(`*logfile: str*`)` → Dict[str, pandas.DataFrame]

Get autometa-large-data-mode-binning runtime information

Data

1. "embedding": Embeddings ranks and times
2. "kmer_count_normalization": K-mer count normalization times
3. "clustering": Embeddings retrieved from cache
4. "skipped_taxa": Ranks above max_partition_size
5. "totals": Total times for all binning tasks

param logfile
Path to autometa-large-data-mode-binning stderr logfile

type logfile
str

returns
Dictionary containing large-data-mode-binning information corresponding to task

rtype
Dict[str, pd.DataFrame]

`autometa.binning.large_data_mode_loginfo.`**`main`**`()`

autometa.binning.recursive_dbscan module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Cluster contigs recursively searching for bins with highest completeness and purity.

`autometa.binning.recursive_dbscan.`**`get_clusters`**(*main: pandas.DataFrame*, *markers_df: pandas.DataFrame*, *completeness: float*, *purity: float*, *coverage_stddev: float*, *gc_content_stddev: float*, *method: str*, *n_jobs: int = -1*, *verbose: bool = False*) → pandas.DataFrame

Find best clusters retained after applying metrics filters.

Parameters

- **`main`** (*`pd.DataFrame`*) – index=contig, cols=['x','y','coverage','gc_content']
- **`markers_df`** (*`pd.DataFrame`*) – wide format, i.e. index=contig cols=[marker,marker,…]
- **`completeness`** (*`float`*) – completeness threshold to retain cluster. e.g. cluster completeness >= completeness
- **`purity`** (*`float`*) – *purity* threshold to retain cluster. e.g. cluster purity >= purity
- **`coverage_stddev`** (*`float`*) – cluster coverage std.dev. threshold to retain cluster. e.g. cluster coverage std.dev. <= coverage_stddev
- **`gc_content_stddev`** (*`float`*) – cluster GC content std.dev. threshold to retain cluster. e.g. cluster GC content std.dev. <= gc_content_stddev
- **`method`** (*`str`*) – Description of parameter *method*. choices = ['dbscan','hdbscan']
- **`verbose`** (*`bool`*) – log stats for each recursive_dbscan clustering iteration

Returns

main with ['cluster','completeness','purity','coverage_stddev','gc_content_stddev'] columns added

Return type

pd.DataFrame

`autometa.binning.recursive_dbscan.`**`main`**`()`

`autometa.binning.recursive_dbscan.`**`recursive_dbscan`**`(`*table: pandas.DataFrame*, *markers_df: pandas.DataFrame*, *completeness_cutoff: float*, *purity_cutoff: float*, *coverage_stddev_cutoff: float*, *gc_content_stddev_cutoff: float*, *n_jobs: int = -1*, *verbose: bool = False*`)` → Tuple[pandas.DataFrame, pandas.DataFrame]

Carry out DBSCAN, starting at eps=0.3 and continuing until there is just one group.

Break conditions to speed up pipeline: Give up if we've got up to eps 1.3 and still no complete and pure clusters. Often when you start at 0.3 there are zero complete and pure clusters, because the groups are too small. Later, some are found as the groups enlarge enough, but after it becomes zero again, it is a lost cause and we may as well stop. On the other hand, sometimes we never find any groups, so perhaps we should give up if by EPS 1.3 we never find any complete/pure groups.

Parameters

- **`table`** (*`pd.DataFrame`*) – Contigs with embedded k-mer frequencies ('x','y'), 'coverage' and 'gc_content' columns
- **`markers_df`** (*`pd.DataFrame`*) – wide format, i.e. index=contig cols=[marker,marker,…]
- **`completeness_cutoff`** (*`float`*) – *completeness_cutoff* threshold to retain cluster (the default is 20.0). e.g. cluster completeness >= completeness_cutoff
- **`purity_cutoff`** (*`float`*) – *purity_cutoff* threshold to retain cluster (the default is 95.0). e.g. cluster purity >= purity_cutoff
- **`coverage_stddev_cutoff`** (*`float`*) – *coverage_stddev_cutoff* threshold to retain cluster (the default is 25.0). e.g. cluster coverage std.dev. <= coverage_stddev_cutoff
- **`gc_content_stddev_cutoff`** (*`float`*) – *gc_content_stddev_cutoff* threshold to retain cluster (the default is 5.0). e.g. cluster gc_content std.dev. <= gc_content_stddev_cutoff
- **`verbose`** (*`bool`*) – log stats for each recursive_dbscan clustering iteration.

Returns

(pd.DataFrame(<passed cutoffs>), pd.DataFrame(<failed cutoffs>)) DataFrames consisting of contigs that passed/failed clustering cutoffs, respectively.

DataFrame:

index = contig columns = ['x,'y','coverage','gc_content','cluster','purity','completeness','coverage_stddev','gc_conte

Return type

2-tuple

`autometa.binning.recursive_dbscan.`**`recursive_hdbscan`**`(`*table: pandas.DataFrame*, *markers_df: pandas.DataFrame*, *completeness_cutoff: float*, *purity_cutoff: float*, *coverage_stddev_cutoff: float*, *gc_content_stddev_cutoff: float*, *n_jobs: int = -1*, *verbose: bool = False*`)` → Tuple[pandas.DataFrame, pandas.DataFrame]

Recursively run HDBSCAN starting with defaults and iterating the min_samples and min_cluster_size until only 1 cluster is recovered.

Parameters

- **`table`** (*`pd.DataFrame`*) – Contigs with embedded k-mer frequencies (‘x’,’y’), ‘coverage’ and ‘gc_content’ columns
- **`markers_df`** (*`pd.DataFrame`*) – wide format, i.e. index=contig cols=[marker,marker,…]
- **`completeness_cutoff`** (*`float`*) – *completeness_cutoff* threshold to retain cluster. e.g. cluster completeness >= completeness_cutoff
- **`purity_cutoff`** (*`float`*) – *purity_cutoff* threshold to retain cluster. e.g. cluster purity >= purity_cutoff
- **`coverage_stddev_cutoff`** (*`float`*) – *coverage_stddev_cutoff* threshold to retain cluster. e.g. cluster coverage std.dev. <= coverage_stddev_cutoff
- **`gc_content_stddev_cutoff`** (*`float`*) – *gc_content_stddev_cutoff* threshold to retain cluster. e.g. cluster gc_content std.dev. <= gc_content_stddev_cutoff
- **`verbose`** (*`bool`*) – log stats for each recursive_dbscan clustering iteration.

Returns

(pd.DataFrame(<passed cutoffs>), pd.DataFrame(<failed cutoffs>)) DataFrames consisting of contigs that passed/failed clustering cutoffs, respectively.

DataFrame:
index = contig columns = [‘x_1’,’x_2’,’coverage’,’gc_content’,’cluster’,’purity’,’completeness’,’coverage_stddev’,’gc_

Return type

2-tuple

`autometa.binning.recursive_dbscan.`**`run_dbscan`**(*df: pandas.DataFrame*, *eps: float*, *n_jobs: int = -1*, *dropcols: List[str] = ['cluster', 'purity', 'completeness', 'coverage_stddev', 'gc_content_stddev']*) → pandas.DataFrame

Run clustering on *df* at provided *eps*.

Notes

- documentation for sklearn DBSCAN
- documentation for HDBSCAN

Parameters

- **`df`** (*`pd.DataFrame`*) – Contigs with embedded k-mer frequencies as [‘x_1’,’x_2’,…, ‘x_ndims’] columns and ‘coverage’ column
- **`eps`** (*`float`*) – The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function. See DBSCAN docs for more details.
- **`dropcols`** (*`list, optional`*) – Drop columns in list from *df* (the default is [‘cluster’,’purity’,’completeness’,’coverage_stddev’,’gc_content_stddev’]).

Returns
df with 'cluster' column added

Return type
pd.DataFrame

Raises
BinningError – Dataframe is missing kmer/coverage annotations

`autometa.binning.recursive_dbscan.`**`run_hdbscan`**(*df: pandas.DataFrame*, *min_cluster_size: int*, *min_samples: int*, *n_jobs: int = -1*) → pandas.DataFrame

Run clustering on *df* at provided *min_cluster_size*.

Notes

- Reasoning for parameter: cluster_selection_method
- Reasoning for parameters: min_cluster_size and min_samples
- Documentation for HDBSCAN

Parameters

- **df** (*pd.DataFrame*) – Contigs with embedded k-mer frequencies as ['x','y'] columns and optionally 'coverage' column
- **min_cluster_size** (*int*) – The minimum size of clusters; single linkage splits that contain fewer points than this will be considered points "falling out" of a cluster rather than a cluster splitting into two new clusters.
- **min_samples** (*int*) – The number of samples in a neighborhood for a point to be considered a core point.
- **n_jobs** (*int*) – Number of parallel jobs to run in core distance computations. For `n_jobs` below -1, (n_cpus + 1 + n_jobs) are used.

Returns
df with 'cluster' column added

Return type
pd.DataFrame

Raises

- **ValueError** – sets *usecols* and *dropcols* may not share elements
- ***TableFormatError*** – *df* is missing k-mer or coverage annotations.

`autometa.binning.recursive_dbscan.`**`taxon_guided_binning`**(*main: pandas.DataFrame*, *markers: pandas.DataFrame*, *completeness: float = 20.0*, *purity: float = 95.0*, *coverage_stddev: float = 25.0*, *gc_content_stddev: float = 5.0*, *starting_rank: str = 'superkingdom'*, *method: str = 'dbscan'*, *reverse_ranks: bool = False*, *n_jobs: int = -1*, *verbose: bool = False*) → pandas.DataFrame

Perform clustering of contigs by provided *method* and use metrics to filter clusters that should be retained via *completeness* and *purity* thresholds.

Parameters

- **main** (*pd.DataFrame*) – index=contig, cols=['x','y', 'coverage', 'gc_content'] taxa cols should be present if *taxonomy* is True. i.e. [taxid,superkingdom,phylum,class,order,family,genus,species]
- **markers** (*pd.DataFrame*) – wide format, i.e. index=contig cols=[marker,marker,…]
- **completeness** (*float, optional*) – Description of parameter *completeness* (the default is 20.).
- **purity** (*float, optional*) – purity threshold to retain cluster (the default is 95.0). e.g. cluster purity >= purity_cutoff
- **coverage_stddev** (*float, optional*) – cluster coverage threshold to retain cluster (the default is 25.0).
- **gc_content_stddev** (*float, optional*) – cluster GC content threshold to retain cluster (the default is 5.0).
- **starting_rank** (*str, optional*) – Starting canonical rank at which to begin subsetting taxonomy (the default is superkingdom). Choices are superkingdom, phylum, class, order, family, genus, species.
- **method** (*str, optional*) – Clustering *method* (the default is 'dbscan'). choices = ['dbscan','hdbscan']
- **reverse_ranks** (*bool, optional*) – False - [superkingdom,phylum,class,order,family,genus,species] (Default) True - [species,genus,family,order,class,phylum,superkingdom]
- **verbose** (*bool, optional*) – log stats for each recursive_dbscan clustering iteration

Returns

main with ['cluster','completeness','purity'] columns added

Return type

pd.DataFrame

Raises

TableFormatError – No marker information is availble for contigs to be binned.

autometa.binning.summary module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Script to summarize Autometa binning results

`autometa.binning.summary.`**`fragmentation_metric`**`(`*df: pandas.DataFrame*, *quality_measure: float = 0.5*`)` → int

Describes the quality of assembled genomes that are fragmented in contigs of different length.

Note: For more information see this metagenomics wiki from Matthias Scholz

Parameters

- **df** (*pd.DataFrame*) – DataFrame to assess fragmentation within metagenome-assembled genome.

- **quality_measure** (*0 < float < 1*) – Description of parameter *quality_measure* (the default is .50). I.e. default measure is N50, but could use .1 for N10 or .9 for N90

Returns
Minimum contig length to cover *quality_measure* of genome (i.e. percentile contig length)

Return type
int

`autometa.binning.summary.`**`get_agg_stats`**(*cluster_groups: pandas.core.groupby.generic.DataFrameGroupBy*, *stat_col: str*) → pandas.DataFrame

Compute min, max, (length weighted) mean and median from provided *stat_col*

Parameters

- **cluster_groups** (*pd.core.groupby.generic.DataFrameGroupBy*) – pandas DataFrame grouped by cluster
- **stat_col** (*str*) – column to on which to compute min, max, (length-weighted) mean and median

Returns
index=cluster, columns=[min_{stat_col}, max_{stat_col}, std_{stat_col}, length_weighted_{stat_col}]

Return type
pd.DataFrame

`autometa.binning.summary.`**`get_metabin_stats`**(*bin_df: pandas.DataFrame*, *markers: Union[str, pandas.DataFrame]*, *cluster_col: str = 'cluster'*) → pandas.DataFrame

Retrieve statistics for all clusters recovered from Autometa binning.

Parameters

- **bin_df** (*pd.DataFrame*) – Autometa binning table. index=contig, cols=['cluster','length', 'gc_content', 'coverage', …]
- **markers** (*str,pd.DataFrame*) – Path to or pd.DataFrame of markers table corresponding to contigs in *bin_df*
- **cluster_col** (*str, optional*) – Clustering column by which to group metabins

Returns
dataframe consisting of various metagenome-assembled genome statistics indexed by cluster.

Return type
pd.DataFrame

Raises

- **TypeError** – markers should be a path to or pd.DataFrame of a markers table corresponding to contigs in *bin_df*
- **ValueError** – One of the required columns (*cluster_col*, coverage, length, gc_content) was not found in *bin_df*

`autometa.binning.summary.`**`get_metabin_taxonomies`**(*bin_df: pandas.DataFrame*, *taxa_db: TaxonomyDatabase*, *cluster_col: str = 'cluster'*) → pandas.DataFrame

Retrieve taxonomies of all clusters recovered from Autometa binning.

Parameters

- **`bin_df`** (*`pd.DataFrame`*) – Autometa binning table. index=contig, cols=['cluster','length','taxid', *canonical_ranks]
- **`taxa_db`** (*`autometa.taxonomy.ncbi.TaxonomyDatabase instance`*) – Autometa NCBI or GTDB class instance
- **`cluster_col`** (*`str, optional`*) – Clustering column by which to group metabins

Returns

Dataframe consisting of cluster taxonomy with taxid and canonical rank. Indexed by cluster

Return type

pd.DataFrame

`autometa.binning.summary.`**`main`**`()`

`autometa.binning.summary.`**`write_cluster_records`**`(`*bin_df: pandas.DataFrame*, *metagenome: str*, *outdir: str*, *cluster_col: str = 'cluster'*`)` → None

Write clusters to *outdir* given clusters *df* and metagenome *records*

Parameters

- **`bin_df`** (*`pd.DataFrame`*) – Autometa binning dataframe. index='contig', cols=['cluster', ...]
- **`metagenome`** (*`str`*) – Path to metagenome fasta file
- **`outdir`** (*`str`*) – Path to output directory to write fastas for each metagenome-assembled genome
- **`cluster_col`** (*`str, optional`*) – Clustering column by which to group metabins

autometa.binning.unclustered_recruitment module

autometa.binning.utilities module

binning utilities script for autometa-binning

Script containing utility functions when performing autometa clustering/classification tasks.

`autometa.binning.utilities.`**`add_metrics`**`(`*df: pandas.DataFrame*, *markers_df: pandas.DataFrame*`)` → Tuple[pandas.DataFrame, pandas.DataFrame]

Adds cluster metrics to each respective contig in df.

- :math:`completeness =

rac{markers_{cluster}}{markers_{ref}} * 100`

- :math:`purity % =

rac{markers_{single-copy}}{markers_{cluster}} * 100`

- :math:`mu_{coverage} =

rac{1}{N}sum_{i=1}^{N}left(x_{i}-mu ight)^{2}`

- :math:`mu_{GC%} =

rac{1}{N}sum_{i=1}^{N}left(x_{i}-mu ight)^{2}`

df
[pd.DataFrame] index='contig' cols=['coverage','gc_content','cluster','x_1','x_2',. . . ,'x_n']

markers_df
[pd.DataFrame] wide format, i.e. index=contig cols=[marker,marker,. . .]

2-tuple
df with added cluster metrics columns=['completeness', 'purity', 'coverage_stddev', 'gc_content_stddev'] pd.DataFrame(index=clusters, cols=['completeness', 'purity', 'coverage_stddev', 'gc_content_stddev'])

`autometa.binning.utilities.`**`apply_binning_metrics_filter`**(*df: pandas.DataFrame*, *completeness_cutoff: float = 20.0*, *purity_cutoff: float = 95.0*, *coverage_stddev_cutoff: float = 25.0*, *gc_content_stddev_cutoff: float = 5.0*) → pandas.DataFrame

Filter *df* by provided cutoff values.

Parameters

- **df** (*pd.DataFrame*) – Dataframe containing binning metrics 'completeness', 'purity', 'coverage_stddev' and 'gc_content_stddev'
- **completeness_cutoff** (*float*) – *completeness_cutoff* threshold to retain cluster (the default is 20.0). e.g. cluster completeness >= completeness_cutoff
- **purity_cutoff** (*float*) – *purity_cutoff* threshold to retain cluster (the default is 95.00). e.g. cluster purity >= purity_cutoff
- **coverage_stddev_cutoff** (*float*) – *coverage_stddev_cutoff* threshold to retain cluster (the default is 25.0). e.g. cluster coverage std.dev. <= coverage_stddev_cutoff
- **gc_content_stddev_cutoff** (*float*) – *gc_content_stddev_cutoff* threshold to retain cluster (the default is 5.0). e.g. cluster gc_content std.dev. <= gc_content_stddev_cutoff

Returns
Cutoff filtered *df*

Return type
pd.DataFrame

Raises
KeyError – One of metrics to apply cutoff does not exist in the *df* columns

`autometa.binning.utilities.`**`filter_taxonomy`**(*df: pandas.DataFrame*, *rank: str*, *name: str*) → pandas.DataFrame

Clean taxon names (by broadcasting lowercase and replacing whitespace) then subset by all contigs under *rank* that are equal to *name*.

Parameters

- **df** (*pd.DataFrame*) – Input dataframe containing columns of canonical ranks.
- **rank** (*str*) – Canonical rank on which to apply filtering.
- **name** (*str*) – Taxon in *rank* to retrieve.

Returns
DataFrame subset by *df[rank] == name*

Return type
pd.DataFrame

Raises

- **`KeyError`** – *rank* not in taxonomy columns.
- **`ValueError`** – Provided *name* not found in *rank* column.

`autometa.binning.utilities.`**`read_annotations`**(*annotations: Iterable*, *how: str = 'inner'*) → pandas.DataFrame

Read in a list of contig annotations from filepaths and return all provided annotations in a single dataframe.

Parameters

- **annotations** (*`Iterable`*) – Filepaths of annotations. These should all contain a 'contig' column to be used as the index
- **how** (*`str, optional`*) – How to join the provided annotations. By default will take the 'inner' or intersection of all contigs from *annotations*.

Returns
index_col='contig', cols=[annotations, …]

Return type
pd.DataFrame

`autometa.binning.utilities.`**`reindex_bin_names`**(*df: pandas.DataFrame*, *cluster_col: str = 'cluster'*, *initial_index: int = 0*) → pandas.DataFrame

Re-index *cluster_col* using the provided *initial_index* as the initial index number then enumerating from this to the number of bins in *cluster_col* of *df*.

Parameters

- **df** (*`pd.DataFrame`*) – Dataframe containing *cluster_col*
- **cluster_col** (*`str, optional`*) – Cluster column to apply reindexing, by default "cluster"
- **initial_index** (*`int, optional`*) – Starting index number when reindexing, by default 0

Note: The bin names will start one number above the *initial_index* number provided. Therefore, the default behavior is to use 0 as the *initial_index* meaning the first bin name will be *bin_1*.

Example

```
>>>import pandas as pd
>>>from autometa.binning.utilities import reindex_bin_names
>>>df = pd.read_csv("binning.tsv", sep='        ', index_col='contig')
>>>reindex_bin_names(df, cluster_col='cluster', initial_index=0)
            cluster  completeness  purity  coverage_stddev  gc_content_stddev
contig
k141_1102126   bin_1     90.647482   100.0          1.20951           1.461658
k141_110415    bin_1     90.647482   100.0          1.20951           1.461658
k141_1210233   bin_1     90.647482   100.0          1.20951           1.461658
k141_1227553   bin_1     90.647482   100.0          1.20951           1.461658
k141_1227735   bin_1     90.647482   100.0          1.20951           1.461658
...              ...           ...     ...              ...                ...
```

(continues on next page)

(continued from previous page)

```
k141_999969      NaN           NaN     NaN               NaN               NaN
k141_99997       NaN           NaN     NaN               NaN               NaN
k141_999982      NaN           NaN     NaN               NaN               NaN
k141_999984      NaN           NaN     NaN               NaN               NaN
k141_999987      NaN           NaN     NaN               NaN               NaN
```

Returns
DataFrame of re-indexed bins in *cluster_col* starting at *initial_index* + 1

Return type
pd.DataFrame

`autometa.binning.utilities.`**`write_results`**(*results: pandas.DataFrame*, *binning_output: str*, *full_output: Optional[str] = None*) → None

Write out binning results with their respective binning metrics

Parameters

- **`results`** (*`pd.DataFrame`*) – Binning results contigs dataframe consisting of “cluster” assignments with their respective metrics and annotations
- **`binning_output`** (*`str`*) – Filepath to write binning results
- **`full_output`** (*`str, optional`*) – If provided, will write assignments, metrics and annotations together into *full_output* (filepath)

Return type
NoneType

`autometa.binning.utilities.`**`zero_pad_bin_names`**(*df: pandas.DataFrame*, *cluster_col: str = 'cluster'*) → pandas.DataFrame

Apply zero padding to *cluster_col* using the length of digit corresponding to the number of unique clusters in *cluster_col* in the *df*.

Parameters

- **`df`** (*`pd.DataFrame`*) – Dataframe containing *cluster_col*
- **`cluster_col`** (*`str, optional`*) – Cluster column to apply zero padding, by default “cluster”

Returns
Dataframe with *cluster_col* zero padded to the length of the number of clusters

Return type
pd.DataFrame

Module contents

autometa.common package

Subpackages

autometa.common.external package

Submodules

autometa.common.external.bedtools module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt. Script containing wrapper functions for bedtools.

`autometa.common.external.bedtools.`**`genomecov`**(*ibam: str*, *out: str*, *force: bool = False*) → str

Run bedtools genomecov with input *ibam* and *lengths* to retrieve metagenome coverages.

Parameters

- **`ibam`** (*`str`*) – </path/to/indexed/BAM/file.ibam>. Note: BAM *must* be sorted by position.
- **`out`** (*`str`*) – </path/to/alignment.bed> The bedtools genomecov output is a tab-delimited file with the following columns: 1. Chromosome 2. Depth of coverage 3. Number of bases on chromosome with that coverage 4. Size of chromosome 5. Fraction of bases on that chromosome with that coverage See also: http://bedtools.readthedocs.org/en/latest/content/tools/genomecov.html
- **`force`** (*`bool`*) – force overwrite of *out* if it already exists (default is False).

Returns

</path/to/alignment.bed>

Return type

str

Raises

- **`FileExistsError`** – *out* file already exists and force is False
- **`OSError`** – Why the exception is raised.

`autometa.common.external.bedtools.`**`main`**()

`autometa.common.external.bedtools.`**`parse`**(*bed: str*, *out: Optional[str] = None*, *force: bool = False*) → pandas.DataFrame

Calculate coverages from bed file.

Parameters

- **`bed`** (*`str`*) – </path/to/file.bed>
- **`out`** (*`str`*) – if provided will write to *out*. I.e. </path/to/coverage.tsv>
- **`force`** (*`bool`*) – force overwrite of *out* if it already exists (default is False).

Returns

index='contig', col='coverage'

Return type
pd.DataFrame

Raises

- `ValueError` – *out* incorrectly formatted to be read as pandas DataFrame.
- `FileNotFoundError` – *bed* does not exist

autometa.common.external.bowtie module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt. Script containing wrapper functions for bowtie2.

`autometa.common.external.bowtie.`**`align`**(*db: str*, *sam: str*, *fwd_reads: Optional[List[str]] = None*, *rev_reads: Optional[List[str]] = None*, *se_reads: Optional[List[str]] = None*, *cpus: int = 0*, ***kwargs*) → str

Align reads to bowtie2-index *db* (at least one **_reads* argument is required).

Parameters

- **db** (*str*) – </path/to/prefix/bowtie2/database>. I.e. *db*.{#}.bt2
- **sam** (*str*) – </path/to/out.sam>
- **fwd_reads** (*list, optional*) – [</path/to/forward_reads.fastq>, …]
- **rev_reads** (*list, optional*) – [</path/to/reverse_reads.fastq>, …]
- **se_reads** (*list, optional*) – [</path/to/single_end_reads.fastq>, …]
- **cpus** (*int, optional*) – Num. processors to use (the default is 0).
- ****kwargs** (*dict, optional*) – Additional optional args to supply to bowtie2. Must be in format: key = flag value = flag-value

Returns
</path/to/out.sam>

Return type
str

Raises
`ChildProcessError` – bowtie2 failed

`autometa.common.external.bowtie.`**`build`**(*assembly: str*, *out: str*) → str

Build bowtie2 index.

Parameters

- **assembly** (*str*) – </path/to/assembly.fasta>
- **out** (*str*) – </path/to/output/database> Note: Indices written will resemble </path/to/output/database.{#}.bt2>

Returns
</path/to/output/database>

Return type
str

Raises
`ChildProcessError` – bowtie2-build failed

`autometa.common.external.bowtie.`**`main`**`()`

`autometa.common.external.bowtie.`**`run`**`(`*cmd: str*`)` → bool

Run *cmd* via subprocess.

Parameters
`cmd` (*`str`*) – Executable input str

Returns
True if no returncode from subprocess.call else False

Return type
bool

autometa.common.external.diamond module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt. Class and functions related to running diamond on metagenome sequences

`autometa.common.external.diamond.`**`blast`**`(`*fasta: str*, *database: str*, *outfpath: str*, *blast_type: str = 'blastp'*, *evalue: float = 1e-05*, *maxtargetseqs: int = 200*, *cpus: int = 2*, *tmpdir: Optional[str] = None*, *force: bool = False*, *verbose: bool = False*`)` → str

Performs diamond blastp search using query sequence against diamond formatted database

Parameters

- **`fasta`** (*`str`*) – Path to fasta file having the query sequences. Should be amino acid sequences in case of BLASTP and nucleotide sequences in case of BLASTX
- **`database`** (*`str`*) – Path to diamond formatted database
- **`outfpath`** (*`str`*) – Path to output file
- **`blast_type`** (*`str, optional`*) – blastp to align protein query sequences against a protein reference database, blastx to align translated DNA query sequences against a protein reference database, by default ‘blastp’
- **`evalue`** (*`float, optional`*) – cutoff e-value to count hit as significant, by default float(‘1e-5’)
- **`maxtargetseqs`** (*`int, optional`*) – max number of target sequences to retrieve per query by diamond, by default 200
- **`cpus`** (*`int, optional`*) – Number of processors to be used, by default uses all the processors of the system
- **`tmpdir`** (*`str, optional`*) – Path to temporary directory. By default, same as the output directory
- **`force`** (*`bool, optional`*) – overwrite existing diamond results, by default False
- **`verbose`** (*`bool, optional`*) – log progress to terminal, by default False

Returns
Path to BLAST results

Return type
str

Raises

- **`FileNotFoundError`** – *fasta* file does not exist
- **`ValueError`** – provided *blast_type* is not ‘blastp’ or ‘blastx’
- **`subprocess.CalledProcessError`** – Failed to run blast

`autometa.common.external.diamond.`**`makedatabase`**(*fasta: str*, *database: str*, *cpus: int = 2*) → str

Creates a database against which the query sequence would be blasted

Parameters

- **`fasta`** (*`str`*) – Path to fasta file whose database needs to be made e.g. ‘<path/to/fasta/file>’
- **`database`** (*`str`*) – Path to the output diamond formatted database file e.g. ‘<path/to/database/file>’
- **`cpus`** (*`int, optional`*) – Number of processors to be used. By default uses all the processors of the system

Returns

Path to diamond formatted database

Return type

str

Raises

`subprocess.CalledProcessError` – Failed to create diamond formatted database

`autometa.common.external.diamond.`**`parse`**(*results: str*, *bitscore_filter: float = 0.9*, *verbose: bool = False*) → Dict[str, Set[str]]

Retrieve diamond results from output table

Parameters

- **`results`** (*`str`*) – Path to BLASTP output file in outfmt6
- **`bitscore_filter`** (*`0 < float <= 1, optional`*) – Bitscore filter applied to each sseqid, by default 0.9 Used to determine whether the bitscore is above a threshold value. For example, if it is 0.9 then only bitscores >= 0.9 * the top bitscore are accepted
- **`verbose`** (*`bool, optional`*) – log progress to terminal, by default False

Returns

{qseqid: {sseqid, sseqid, … }, … }

Return type

dict

Raises

- **`FileNotFoundError`** – diamond results table does not exist
- **`ValueError`** – bitscore_filter value is not a float or not in range of 0 to 1

autometa.common.external.hmmscan module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt. Functions related to running hmmer on metagenome sequences

`autometa.common.external.hmmscan.`**`annotate_parallel`**(*orfs*, *hmmdb*, *outfpath*, *cpus*, *seed=42*)

`autometa.common.external.hmmscan.`**`annotate_sequential`**(*orfs*, *hmmdb*, *outfpath*, *cpus*, *seed=42*)

`autometa.common.external.hmmscan.`**`filter_tblout_markers`**(*infpath: str*, *cutoffs: str*, *outfpath: Optional[str] = None*, *orfs: Optional[str] = None*, *force: bool = False*) → pandas.DataFrame

Filter markers from hmmscan tblout output table using provided cutoff values file.

Parameters

- **`infpath`** (*`str`*) – Path to hmmscan tblout output file
- **`cutoffs`** (*`str`*) – Path to marker set inclusion cutoffs
- **`outfpath`** (*`str, optional`*) – Path to write filtered markers to tab-delimited file
- **`orfs`** (*`str, optional`*) – Default will attempt to translate recovered qseqids to contigs </path/to/prodigal/called/orfs.fasta>
- **`force`** (*`bool, optional`*) – Overwrite existing *outfpath* (the default is False).

Returns

</path/to/output.markers.tsv>

Return type

pd.DataFrame

Raises

- **`FileNotFoundError`** – *infpath* or *cutoffs* not found
- **`FileExistsError`** – *outfpath* already exists and force=False
- **`AssertionError`** – No returned markers pass the cutoff thresholds. I.e. final df is empty.

`autometa.common.external.hmmscan.`**`hmmpress`**(*fpath*)

Runs hmmpress on *fpath*.

Parameters

`fpath` (*`str`*) – </path/to/kindom.markers.hmm>

Returns

</path/to/hmmpressed/kindom.markers.hmm>

Return type

str

Raises

- **`FileNotFoundError`** – *fpath* not found.
- **`subprocess.CalledProcessError`** – hmmpress failed

`autometa.common.external.hmmscan.`**`main`**()

`autometa.common.external.hmmscan.`**`read_domtblout`**(*fpath: str*) → pandas.DataFrame

Read hmmscan domtblout-format results into pandas DataFrame

For more detailed column descriptions see the ‘tabular output formats’ section in the [HMMER manual](http://eddylab.org/software/hmmer/Userguide.pdf#tabular-output-formats “HMMER Manual”)

Parameters

`fpath` (*`str`*) – Path to hmmscan domtblout file

Returns

index=range(0,n_hits) cols=…

Return type

pd.DataFrame

`autometa.common.external.hmmscan.`**`read_tblout`**(*infpath: str*) → pandas.DataFrame

Read hmmscan tblout-format results into pd.DataFrame

For more detailed column descriptions see the ‘tabular output formats’ section in the [HMMER manual](http://eddylab.org/software/hmmer/Userguide.pdf#tabular-output-formats “HMMER Manual”)

Parameters

`infpath` (*`str`*) – Path to hmmscan_results.tblout

Returns

DataFrame of raw hmmscan results

Return type

pd.DataFrame

Raises

`FileNotFoundError` – Path to *infpath* was not found

`autometa.common.external.hmmscan.`**`run`**(*orfs*, *hmmdb*, *outfpath*, *cpus=0*, *force=False*, *parallel=True*, *gnu_parallel=False*, *seed=42*) → str

Runs hmmscan on dataset ORFs and provided hmm database.

Note: Only one of *parallel* and *gnu_parallel* may be provided as True

Parameters

- **`orfs`** (*`str`*) – </path/to/orfs.faa>
- **`hmmdb`** (*`str`*) – </path/to/hmmpressed/database.hmm>
- **`outfpath`** (*`str`*) – </path/to/output.hmmscan.tsv>
- **`cpus`** (*`int, optional`*) – Num. cpus to use. 0 will run as many cpus as possible (the default is 0).
- **`force`** (*`bool, optional`*) – Overwrite existing *outfpath* (the default is False).
- **`parallel`** (*`bool, optional`*) – Will use multithreaded parallelization offered by hmmscan (the default is True).
- **`gnu_parallel`** (*`bool, optional`*) – Will parallelize hmmscan using GNU parallel (the default is False).
- **`seed`** (*`int, optional`*) – set RNG seed to <n> (if 0: one-time arbitrary seed) (the default is 42).

Returns
</path/to/output.hmmscan.tsv>

Return type
str

Raises

- **`ValueError`** – Both *parallel* and *gnu_parallel* were provided as True
- **`FileExistsError`** – *outfpath* already exists
- **`subprocess.CalledProcessError`** – hmmscan failed

autometa.common.external.hmmsearch module

Module to filter the domtbl file from hmmsearch –domtblout <filepath> using provided cutoffs

`autometa.common.external.hmmsearch.`**`filter_domtblout`**`(`*infpath: str*, *cutoffs: str*, *orfs: str*, *outfpath: Optional[str] = None*`)` → pandas.DataFrame

`autometa.common.external.hmmsearch.`**`main`**`()`

autometa.common.external.prodigal module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Functions to retrieve orfs from provided assembly using prodigal

`autometa.common.external.prodigal.`**`aggregate_orfs`**`(`*search_str: str*, *outfpath: str*`)` → None

`autometa.common.external.prodigal.`**`annotate_parallel`**`(`*assembly: str*, *prots_out: str*, *nucls_out: str*, *cpus: int*`)` → None

`autometa.common.external.prodigal.`**`annotate_sequential`**`(`*assembly: str*, *prots_out: str*, *nucls_out: str*`)` → None

`autometa.common.external.prodigal.`**`contigs_from_headers`**`(`*fpath: str*`)` → Mapping[str, str]

Get ORF id to contig id translations using prodigal assigned ID from description.

First determines if all of ID=3495691_2 from description is in header. “3495691_2” represents the 3,495,691st gene in the 2nd sequence.

Example

```
#: prodigal versions < 2.6 record
>>>record.id
'k119_1383959_3495691_2'

>>>record.description
'k119_1383959_3495691_2 # 688 # 1446 # 1 # ID=3495691_2;partial=01;start_type=ATG;
↪rbs_motif=None;rbs_spacer=None'

>>>record.description.split('#')[-1].split(';')[0].strip()
```

(continues on next page)

(continued from previous page)

```
'ID=3495691_2'

>>>orf_id = '3495691_2'
'3495691_2'

>>>record.id.replace(f'_{orf_id}', '')
'k119_1383959'

#: prodigal versions >= 2.6 record
>>>record.id
'k119_1383959_2'
>>>record.id.rsplit('_',1)[0]
'k119_1383959'
```

Parameters
`fpath` (*str*) – </path/to/prodigal/called/orfs.fasta>

Returns
contigs translated from prodigal ORF description. {orf_id:contig_id, . . . }

Return type
dict

`autometa.common.external.prodigal.`**`main`**`()`

`autometa.common.external.prodigal.`**`orf_records_from_contigs`**`(`*contigs: Union[List, Set]*, *fpath: str*`)` → List[Bio.SeqIO.SeqRecord]

Retrieve list of *ORFs headers* from *contigs*. Prodigal annotated ORFs are required as the input *fpath*.

Parameters

- **`contigs`** (*iterable*) – iterable of contigs from which to retrieve ORFs
- **`fpath`** (*str*) – </path/to/prodigal/called/orfs.fasta>

Returns
ORF SeqIO.SeqRecords from provided *contigs*. i.e. [SeqRecord, . . .]

Return type
list

Raises
`ExceptionName` – Why the exception is raised.

`autometa.common.external.prodigal.`**`run`**`(`*assembly: str*, *nucls_out: str*, *prots_out: str*, *force: bool = False*, *cpus: int = 0*`)` → Tuple[str, str]

Calls ORFs from provided input assembly

Parameters

- **`assembly`** (*str*) – </path/to/assembly.fasta>
- **`nucls_out`** (*str*) – </path/to/nucls.out>
- **`prots_out`** (*str*) – </path/to/prots.out>
- **`force`** (*bool*) – overwrite outfpath if it already exists (the default is False).
- **`cpus`** (*int*) – num *cpus* to use. **Default (cpus=0) will run as many `cpus` as possible**

Returns
(*nucls_out*, *prots_out*)

Return type
2-Tuple

Raises

- **`FileExistsError`** – *nucls_out* or *prots_out* already exists
- **`subprocess.CalledProcessError`** – prodigal Failed
- **`ChildProcessError`** – *nucls_out* or *prots_out* not written
- **`IOError`** – *nucls_out* or *prots_out* incorrectly formatted

autometa.common.external.samtools module

Script containing wrapper functions for samtools

`autometa.common.external.samtools.`**`main`**`()`

`autometa.common.external.samtools.`**`sort`**`(`*sam*, *bam*, *cpus=2*`)`

Views then sorts sam file by leftmost coordinates and outputs to bam.

Parameters

- **sam** (*`str`*) – </path/to/alignment.sam>
- **bam** (*`str`*) – </path/to/output/alignment.bam>
- **cpus** (*`int, optional`*) – Number of processors to be used. By default uses all the processors of the system

Raises

- **`TypeError`** – cpus must be an integer greater than zero
- **`FileNotFoundError`** – Specified path is incorrect or the file is empty
- *`ExternalToolError`* – Samtools did not run successfully, returns subprocess traceback and command run

Module contents

Submodules

autometa.common.coverage module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Calculates coverage of contigs

`autometa.common.coverage.`**`from_spades_names`**`(`*records: List[Bio.SeqRecord.SeqRecord]*`)` → pandas.DataFrame

Retrieve coverages from SPAdes scaffolds headers.

Example SPAdes header : NODE_83_length_162517_cov_224.639

Parameters

`records` (*List[SeqRecord]*) – [SeqRecord,. . .]

Returns

index=contig, name='coverage', dtype=float

Return type

pd.DataFrame

`autometa.common.coverage.get`(*fasta: str*, *out: str*, *from_spades: bool = False*, *fwd_reads: List[str] = None*, *rev_reads: List[str] = None*, *se_reads: List[str] = None*, *sam: str = None*, *bam: str = None*, *bed: str = None*, *cpus: int = 1*) → pandas.DataFrame

Get coverages for assembly *fasta* file using provided files or if the metagenome assembly was generated from SPAdes, use the k-mer coverages provided in each contig's header by specifying *from_spades=True*.

Either *fwd_reads* and *rev_reads* and/or *se_reads* or,`sam`, or *bam*, or *bed* must be provided if *from_spades=False*.

Note: Will begin coverage calculation based on files provided checking in the following order:

1. *bed*
2. *bam*
3. *sam*
4. *fwd_reads* and *rev_reads* and *se_reads*

Event sequence to calculate contig coverages:

1. align reads to generate alignment.sam
2. sort samfile to generate alignment.bam
3. calculate assembly coverages to generate alignment.bed
4. calculate contig coverages to generate coverage.tsv

Parameters

- **fasta** (*str*) – </path/to/assembly.fasta>
- **out** (*str*) – </path/to/output/coverages.tsv>
- **from_spades** (*bool, optional*) – If True, will attempt to parse record ids for coverage information. This is only compatible with SPAdes assemblies. (the Default is False).
- **fwd_reads** (*List[str], optional*) – [</path/to/forward_reads.fastq>, . . .]
- **rev_reads** (*List[str], optional*) – [</path/to/reverse_reads.fastq>, . . .]
- **se_reads** (*List[str], optional*) – [</path/to/single_end_reads.fastq>, . . .]
- **sam** (*str, optional*) – </path/to/alignments.sam>
- **bam** (*str, optional*) – </path/to/alignments.bam>
- **bed** (*str, optional*) – </path/to/alignments.bed>
- **cpus** (*int, optional*) – Number of cpus to use for coverage calculation.

Returns

index=contig cols=['coverage']

Return type

pd.DataFrame

`autometa.common.coverage.`**`main`**`()`

autometa.common.exceptions module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

File containing customized AutometaErrors for more specific exception handling

`exception` `autometa.common.exceptions.`**`AutometaError`**

Bases: `Exception`

Base class for Autometa Errors.

`exception` `autometa.common.exceptions.`**`BinningError`**

Bases: *`AutometaError`*

BinningError exception class.

Exception called when issues arise during or after the binning process.

This is usually a result of no clusters being recovered.

`exception` `autometa.common.exceptions.`**`ChecksumMismatchError`**

Bases: *`AutometaError`*

ChecksumMismatchError exception class

Exception called when checksums do not match.

`exception` `autometa.common.exceptions.`**`DatabaseOutOfSyncError`**`(`*value*`)`

Bases: *`AutometaError`*

Raised when NCBI databases nodes.dmp, names.dmp and merged.dmp are out of sync with each other :param AutometaError: Base class for other exceptions :type AutometaError: class

`__str__``()`

Operator overloading to return the text message written while raising the error, rather than the message of __str__ by base exception :returns: Message written alongside raising the exception :rtype: str

`exception` `autometa.common.exceptions.`**`ExternalToolError`**`(`*cmd*, *err*`)`

Bases: *`AutometaError`*

Raised when samtools sort is not executed properly.

Parameters

`AutometaError` (*`class`*) – Base class for other exceptions

`exception` `autometa.common.exceptions.`**`TableFormatError`**

Bases: *`AutometaError`*

TableFormatError exception class.

Exception called when Table format is incorrect.

This is usually a result of a table missing the ‘contig’ column as this is often used as the index.

autometa.common.kmers module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Count, normalize and embed k-mers given nucleotide sequences

`autometa.common.kmers.`**`autometa_clr`**`(`*df: pandas.DataFrame*`)` → pandas.DataFrame

Normalize k-mers by Centered Log Ratio transformation

Steps

- Drop any k-mers not present for all contigs
- Drop any contigs not containing any kmer counts
- Fill any remaining na values with 0
- Normalize the k-mer count by the total count of all k-mers for a given contig
- Add 1 as 0 can not be utilized for CLR
- Perform CLR transformation log(norm. value / geometric mean norm. value)

param df
K-mers Dataframe where index_col='contig' and column values are k-mer frequencies.

type df
pd.DataFrame

References

- Aitchison, J. The Statistical Analysis of Compositional Data (1986)
- Pawlowsky-Glahn, Egozcue, Tolosana-Delgado. Lecture Notes on Compositional Data Analysis (2011)
- Why ILR is preferred stats stackexchange discussion
- Use of CLR transformation prior to PCA stats stackexchange discussion
- Lecture notes on Compositional Data Analysis (CoDa) PDF

returns
index='contig', cols=[kmer, kmer, …] Columns have been transformed by CLR normalization.

rtype
pd.DataFrame

`autometa.common.kmers.`**`count`**`(`*assembly: str*, *size: int = 5*, *out: str = None*, *force: bool = False*, *verbose: bool = True*, *cpus: int = 2*`)` → pandas.DataFrame

Counts k-mer frequencies for provided assembly file

First we make a dictionary of all the possible k-mers (discounting reverse complements). Each k-mer's count is updated by index when encountered in the record.

Parameters

- **`assembly`** (*`str`*) – Description of parameter *assembly*.
- **`size`** (*`int, optional`*) – length of k-mer to count *size* (the default is 5).

- **out** (*str, optional*) – Path to write k-mer counts table.
- **force** (*bool, optional*) – Whether to overwrite existing *out* k-mer counts table (the default is False).
- **verbose** (*bool, optional*) – Enable progress bar *verbose* (the default is True).
- **cpus** (*int, optional*) – Number of cpus to use. (the default will use all available).

Returns

index_col='contig', tab-delimited, cols=unique_kmers i.e. 5-mer columns=[AAAAA, AAAAT, AAAAC, AAAAG, …, GCCGC]

Return type

pandas.DataFrames

Raises

TypeError – *size* must be an int

`autometa.common.kmers.`**`embed`**(*kmers: Union[str, pandas.DataFrame]*, *out: Optional[str] = None*, *force: bool = False*, *embed_dimensions: int = 2*, *pca_dimensions: int = 50*, *method: str = 'bhsne'*, *perplexity: float = 30.0*, *seed: int = 42*, *n_jobs: int = -1*, ***method_kwargs: Dict[str, Any]*) → pandas.DataFrame

Embed k-mers using provided *method*.

Notes

- sklearn.manifold.TSNE
- tsne.bh_sne
- UMAP
- densMAP
- TriMap

Parameters

- **kmers** (*str or pd.DataFrame*) – </path/to/input/kmers.normalized.tsv>
- **out** (*str, optional*) – </path/to/output/kmers.out.tsv> If provided will write to *out*.
- **force** (*bool, optional*) – Whether to overwrite existing *out* file.
- **embed_dimensions** (*int, optional*) – embed_dimensions` to embed k-mer frequencies (the default is 2).

  The output embedded kmers will follow columns of *x_1* to *x_{embed_dimensions}*

  NOTE: The columns are 1-indexed, i.e. at x_1 *not* x_0
- **pca_dimensions** (*int, optional*) – Reduce k-mer frequencies dimensions to *pca_dimensions* (the default is 50). If zero, will skip this step.
- **method** (*str, optional*) – embedding method to use (the default is 'bhsne'). choices include sksne, bhsne, umap, trimap and densmap.
- **perplexity** (*float, optional*) – hyperparameter used to tune sksne and bhsne (the default is 30.0).
- **seed** (*int, optional*) – Seed to use for *method*. Allows for reproducibility from random state.

- **n_jobs** (*int, optional*) – Used with *sksne*, *densmap* and *umap*, (the default is -1 which will attempt to use all available CPUs)

Note: For n_jobs below -1, (CPUS + 1 + n_jobs) are used. For example with n_jobs=-2, all CPUs but one are used.

- scikit-learn TSNE n_jobs glossary
- UMAP and DensMAP's

invocation use this with pynndescent

**method_kwargs : Dict[str, Any], optional

Other keyword arguments (kwargs) to be supplied to respective *method*.

Set UMAP(verbose=True, output_dens=True) using **method_kwargs >>> embed_df = kmers.embed(

norm_df, method='densmap', embed_dimensions=2, n_jobs=None, **{

'verbose': True, 'output_dens': True,

}

)

NOTE: Setting duplicate arguments will result in an error

Here we specify `UMAP(densmap=True)` using `method='densmap'` and also attempt to overwrite to `UMAP(densmap=False)` with the method_kwargs, `**{'densmap':False}`, resulting in a TypeError.

```
>>> embed_df = kmers.embed(
    df,
    method='densmap',
    embed_dimensions=2,
    n_jobs=4,
    **{'densmap': False}
)
TypeError: umap.umap_.UMAP() got multiple values for keyword argument
↪'densmap'
```

Typically, you will not require the use of method_kwargs as this is only available for applying advanced parameter settings to any of the available embedding methods.

Returns

out dataframe with index='contig' and cols=['x','y','z']

Return type

pd.DataFrame

Raises

- **TypeError** – Provided *kmers* is not a str or pd.DataFrame.
- ***TableFormatError*** – Provided *kmers* or *out* are not formatted correctly for use.
- **ValueError** – Provided *method* is not an available choice.
- **FileNotFoundError** – *kmers* type must be a pd.DataFrame or filepath.

`autometa.common.kmers.`**`init_kmers`**`(`*kmer_size: int = 5*`)` → Dict[str, int]

Initialize k-mers from *kmer_size*. Respective reverse complements will be removed.

Parameters
`kmer_size` (*`int, optional`*) – pattern size of k-mer to intialize dict (the default is 5).

Returns
{kmer:index, . . . }

Return type
dict

`autometa.common.kmers.`**`load`**`(`*kmers_fpath: str*`)` → pandas.DataFrame

Load in a previously counted k-mer frequencies table.

Parameters
`kmers_fpath` (*`str`*) – Path to kmer frequency table

Returns
index='contig', cols=[kmer, kmer, . . .]

Return type
pd.DataFrame

Raises

- **`FileNotFoundError`** – *kmers_fpath* does not exist or is empty
- *`TableFormatError`* – *kmers_fpath* file format is invalid

`autometa.common.kmers.`**`main`**`()`

`autometa.common.kmers.`**`mp_counter`**`(`*assembly: str*, *ref_kmers: Dict[str, int]*, *cpus: int = 2*`)` → List

Multiprocessing k-mer counter used in *count*. (Should not be used directly).

Parameters

- **`assembly`** (*`str`*) – </path/to/assembly.fasta> (nucleotides)
- **`ref_kmers`** (*`dict`*) – {kmer:index, . . . }
- **`cpus`** (*`int, optional`*) – Number of cpus to use. (the default will use all available).

Returns
[{record:counts}, {record:counts}, . . .]

Return type
list

`autometa.common.kmers.`**`normalize`**`(`*df: pandas.DataFrame*, *method: str = 'am_clr'*, *out: Optional[str] = None*, *force: bool = False*`)` → pandas.DataFrame

Normalize raw k-mer counts by center or isometric log-ratio transform.

Parameters

- **`df`** (*`pd.DataFrame`*) – k-mer counts dataframe. i.e. for 3-mers; Index='contig', columns=[AAA, AAT, . . .]
- **`method`** (*`str, optional`*) – Normalize k-mer counts using CLR or ILR transformation (the default is Autometa's CLR implementation). choices = ['ilr', 'clr', 'am_clr'] Other transformations come from the skbio.stats.composition module
- **`out`** (*`str, optional`*) – Path to write normalized k-mers.
- **`force`** (*`bool, optional`*) – Whether to overwrite existing *out* file path, by default False.

Returns
Normalized counts using provided *method*.

Return type
pd.DataFrame

Raises
`ValueError` – Provided *method* is not available.

`autometa.common.kmers.`**`record_counter`**`(`*args: Tuple[Bio.SeqIO.SeqRecord, Dict[str, int]]*`)` → Dict[str, List[int]]

single record counter used when multiprocessing.

Parameters
`args` (*`2-tuple`*) – (record, ref_kmers) - record : SeqIO.SeqRecord - ref_kmers : {kmer:index, . . . }

Returns
{contig:[count,count,. . .]} count index is respective to ref_kmers.keys()

Return type
dict

`autometa.common.kmers.`**`seq_counter`**`(`*assembly: str*, *ref_kmers: Dict[str, int]*, *verbose: bool = True*`)` → Dict[str, List[int]]

Sequentially count k-mer frequencies.

Parameters

- **`assembly`** (*`str`*) – </path/to/assembly.fasta> (nucleotides)
- **`ref_kmers`** (*`dict`*) – {kmer:index, . . . }
- **`verbose`** (*`bool, optional`*) – enable progress bar *verbose* (the default is True).

Returns
{contig:[count,count,. . .]} count index is respective to ref_kmers.keys()

Return type
dict

autometa.common.markers module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Autometa Marker class consisting of various methods to annotate sequences with marker sets depending on sequence set taxonomy

`autometa.common.markers.`**`get`**`(`*kingdom: str*, *orfs: str*, *hmmdb: Optional[str] = None*, *cutoffs: Optional[str] = None*, *dbdir: str = './autometa/databases/markers'*, *scans: Optional[str] = None*, *out: Optional[str] = None*, *force: bool = False*, *cpus: int = 8*, *parallel: bool = True*, *gnu_parallel: bool = False*, *seed: int = 42*`)` → pandas.DataFrame

Retrieve contigs' markers from markers database that pass cutoffs filter.

Parameters

- **`kingdom`** (*`str`*) – kingdom to annotate markers choices = ['bacteria', 'archaea']
- **`orfs`** (*`str`*) – Path to amino-acid ORFs file

- `dbdir` – Optional directory containing hmmdb and cutoffs files
- `hmmdb` – Path to marker genes database file, previously hmmpressed.
- `cutoffs` – Path to marker genes cutoff tsv.
- `scans` (*str, optional*) – Path to existing hmmscan table to filter by cutoffs
- `out` (*str, optional*) – Path to write annotated markers table.
- `force` (*bool, optional*) – Whether to overwrite existing *out* file path, by default False.
- `cpus` (*int, optional*) – Number of cores to use if running in parallel, by default all available.
- `parallel` (*bool, optional*) – Whether to run hmmscan using its parallel option, by default True.
- `gnu_parallel` (*bool, optional*) – Whether to run hmmscan using gnu parallel, by default False.
- `seed` (*int, optional*) – Seed to pass into hmmscan for determinism, by default 42.

Returns

- wide - pd.DataFrame(index_col=contig, columns=[PFAM,…])
- long - pd.DataFrame(index_col=contig, columns=['sacc','count'])
- list - {contig:[pfam,pfam,…],contig:[…],…}
- counts - {contig:count, contig:count,…}

Return type

pd.Dataframe or dict

Raises

`ValueError` – Why the exception is raised.

`autometa.common.markers.`**`load`**(*fpath*, *format='wide'*)

Read markers table into specified *format*.

Parameters

- `fpath` (*str*) – </path/to/*kingdom*.markers.tsv>
- `format` (*str, optional*) –
  - wide - index=contig, cols=[domain sacc,..] (default)
  - long - index=contig, cols=['sacc','count']
  - list - {contig:[sacc,…],…}
  - counts - {contig:len([sacc,…]), …}

Returns

- wide - index=contig, cols=[domain sacc,..] (default)
- long - index=contig, cols=['sacc','count']
- list - {contig:[sacc,…],…}
- counts - {contig:len([sacc,…]), …}

Return type

pd.DataFrame or dict

Raises

- **`FileNotFoundError`** – Provided *fpath* does not exist
- **`ValueError`** – Provided *format* is not in choices: choices = wide, long, list or counts

`autometa.common.markers.`**`main`**`()`

autometa.common.metagenome module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Script containing Metagenome class for general handling of metagenome assembly

`class` `autometa.common.metagenome.`**`Metagenome`**`(`*assembly*`)`

Bases: `object`

Autometa Metagenome Class.

Parameters
`assembly` (*`str`*) – </path/to/metagenome/assembly.fasta>

`sequences`

[seq,. . .]

Type
list

`seqrecords`

[SeqRecord,. . .]

Type
list

`nseqs`

Number of sequences in assembly.

Type
int

`length_weighted_gc`

Length weighted average GC% of assembly.

Type
float

`size`

Total assembly size in bp.

Type
int

`largest_seq`

id of longest sequence in assembly

Type
str

`\* self.fragmentation_metric()`

* `self.describe()`

* `self.length_filter()`

describe() → pandas.DataFrame

Return dataframe of details.

Columns

# assembly : Assembly input into Metagenome(...) [index column] # nseqs : Number of sequences in assembly # size : Size or total sum of all sequence lengths # N50 : # N10 : # N90 : # length_weighted_gc_content : Length weighted average GC content # largest_seq : Largest sequence in assembly

rtype

pd.DataFrame

fragmentation_metric(*quality_measure: float = 0.5*) → int

Describes the quality of assembled genomes that are fragmented in contigs of different length.

Note: For more information see this metagenomics wiki from Matthias Scholz

---

Parameters

quality_measure (`0 < float < 1`) – Description of parameter *quality_measure* (the default is .50). I.e. default measure is N50, but could use .1 for N10 or .9 for N90

Returns

Minimum contig length to cover *quality_measure* of genome (i.e. length weighted median)

Return type

int

gc_content() → pandas.DataFrame

Retrieves GC content from sequences in assembly

Returns

index="contig", columns=["gc_content","length"]

Return type

pd.DataFrame

property largest_seq: str

Retrieve the name of the largest sequence in the provided *assembly*.

Returns

record ID of the largest sequence in *assembly*.

Return type

str

length_filter(*out: str*, *cutoff: int = 3000*, *force: bool = False*)

Filters sequences by length with provided cutoff.

Note: A WARNING will be emitted and the original metagenome will be returned if no contigs pass the length filter cutoff.

Parameters

- **`out`** (*`str`*) – Path to write length filtered output fasta file.
- **`cutoff`** (*`int, optional`*) – Lengths above or equal to *cutoff* that will be retained (the default is 3000).
- **`force`** (*`bool, optional`*) – Overwrite existing *out* file (the default is False).

Returns
autometa Metagenome object with only assembly sequences above the cutoff threshold.

Return type
Metagenome

Raises

- **`TypeError`** – cutoff value must be a float or integer
- **`ValueError`** – cutoff value must be a positive real number
- **`FileExistsError`** – filepath consisting of sequences that passed filter already exists

`property length_weighted_gc: float`

Retrieve the length weighted average GC percentage of provided *assembly*.

Returns
GC percentage weighted by contig length.

Return type
float

`property nseqs: int`

Retrieve the number of sequences in provided *assembly*.

Returns
Number of sequences parsed from *assembly*

Return type
int

`property seqrecords: list`

Retrieve SeqRecord objects from provided *assembly*.

Returns
[SeqRecord, SeqRecord, …]

Return type
list

`property sequences: list`

Retrieve the sequences from provided *assembly*.

Returns
[seq, seq, …]

Return type
list

property size: int

Retrieve the summation of sizes for each contig in the provided *assembly*.

Returns
Total summation of contig sizes in *assembly*

Return type
int

`autometa.common.metagenome.`**`main`**`()`

autometa.common.utilities module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

File containing common utilities functions to be used by Autometa scripts.

`autometa.common.utilities.`**`calc_checksum`**`(`*fpath: str*`)` → str

Retrieve md5 checksum from provided *fpath*.

See:
https://stackoverflow.com/questions/3431825/generating-an-md5-checksum-of-a-file

fpath
[str] </path/to/file>

str
space-delimited hexdigest of *fpath* using md5sum and basename of *fpath*. e.g. 'hash filename

'

FileNotFoundError
Provided *fpath* does not exist

TypeError
fpath is not a string

`autometa.common.utilities.`**`file_length`**`(`*fpath: str*, *approximate: bool = False*`)` → int

Retrieve the number of lines in *fpath*

See: https://stackoverflow.com/q/845058/13118765

Parameters

- **`fpath`** (*`str`*) – Description of parameter *fpath*.
- **`approximate`** (*`bool`*) – If True, will approximate the length of the file from the file size.

Returns
Number of lines in *fpath*

Return type
int

Raises
`FileNotFoundError` – provided *fpath* does not exist

`autometa.common.utilities.`**`gunzip`**`(`*infpath: str*, *outfpath: str*, *delete_original: bool = False*, *block_size: int = 65536*`)` → str

Decompress gzipped *infpath* to *outfpath* and write checksum of *outfpath* upon successful decompression.

Parameters

- **`infpath`** (*`str`*) – </path/to/file.gz>
- **`outfpath`** (*`str`*) – </path/to/file>
- **`delete_original`** (*`bool`*) – Will delete the original file after successfully decompressing *infpath* (Default is False).
- **`block_size`** (*`int`*) – Amount of *infpath* to read in to memory before writing to *outfpath* (Default is 65536 bytes).

Returns
</path/to/file>

Return type
str

Raises
`FileExistsError` – *outfpath* already exists and is not empty

`autometa.common.utilities.`**`internet_is_connected`**`(`*host: str = '8.8.8.8'*, *port: int = 53*, *timeout: int = 2*`)` → bool

`autometa.common.utilities.`**`is_gz_file`**`(`*filepath*`)` → bool

Check if the given file is gzipped compressed or not.

Parameters
`filepath` (*`str`*) – Filepath to check

Returns
True if file is gzipped else False

Return type
bool

`autometa.common.utilities.`**`make_pickle`**`(`*obj: Any*, *outfpath: str*`)` → str

Serialize a python object (*obj*) to *outfpath*. Note: Opposite of `unpickle()`

Parameters

- **`obj`** (*`any`*) – Python object to serialize to outfpath.
- **`outfpath`** (*`str`*) – </path/to/pickled/file>.

Returns
</path/to/pickled/file.pkl>

Return type
str

Raises
`ExceptionName` – Why the exception is raised.

`autometa.common.utilities.`**`ncbi_is_connected`**`(`*filepath: str = 'rsync://ftp.ncbi.nlm.nih.gov/genbank/GB_Release_Number'*`)` → bool

Check if ncbi databases are reachable. This can be used instead of a check for internet connection.

Parameters

- **`filepath`** (*`string`*) – filepath to NCBI’s rsync server. Default is rsync://ftp.ncbi.nlm.nih.gov/genbank/GB_Release_Number, which should be a very small file that is unlikely to move. This may need to be updated if NCBI changes their file organization.
- **`Outputs`** –
- **`-------`** –
- **`False`** (*`True or`*) – True if the rsync server can be contacted without an error False if the rsync process returns any error

`autometa.common.utilities.`**`read_checksum`**(*fpath: str*) → str

Read checksum from provided checksum formatted *fpath*.

Note: See *write_checksum* for how a checksum file is generated.

Parameters
`fpath` (*`str`*) – </path/to/file.md5>

Returns
checksum retrieved from *fpath*.

Return type
str

Raises

- **`TypeError`** – Provided *fpath* was not a string.
- **`FileNotFoundError`** – Provided *fpath* does not exist.

`autometa.common.utilities.`**`tarchive_results`**(*outfpath: str*, *src_dirpath: str*) → str

Generate a tar archive of Autometa Results

See: https://stackoverflow.com/questions/2032403/how-to-create-full-compressed-tar-file-using-python

Parameters

- **`outfpath`** (*`str`*) – </path/to/output/tar/archive.tar.gz || </path/to/output/tar/archive.tgz
- **`src_dirpath`** (*`str`*) – </paths/to/directory/to/archive>

Returns
</path/to/output/tar/archive.tar.gz || </path/to/output/tar/archive.tgz

Return type
str

Raises
`FileExistsError` – *outfpath* already exists

`autometa.common.utilities.`**`timeit`**(*func: function*) → function

Time function run time (to be used as a decorator). I.e. when defining a function use python’s decorator syntax

Example

```
@timeit
def your_function(args):
    ...
```

Notes

See: https://docs.python.org/2/library/functools.html#functools.wraps

Parameters
func (*function*) – function to decorate timer

Returns
timer decorated *func*.

Return type
function

`autometa.common.utilities.`**`unpickle`**(*fpath: str*) → Any

Load a serialized *fpath* from `make_pickle()`.

Parameters
fpath (*str*) – </path/to/file.pkl>.

Returns
Python object that was serialized to file via make_pickle()

Return type
any

Raises
ExceptionName – Why the exception is raised.

`autometa.common.utilities.`**`untar`**(*tarchive: str*, *outdir: str*, *member: Optional[str] = None*) → str

Decompress a tar archive (may be gzipped or bzipped). passing in *member* requires an *outdir* also be provided.

See: https://docs.python.org/3.8/library/tarfile.html#module-tarfile

Parameters

- **tarchive** (*str*) – </path/tarchive.tar.[compression]>
- **outdir** (*str*) – </path/to/output/directory>
- **member** (*str, optional*) – member file to extract.

Returns
</path/to/extracted/member/file> if member else </path/to/output/directory>

Return type
str

Raises

- **IsADirectoryError** – *outdir* already exists
- **ValueError** – *tarchive* is not a tar archive
- **KeyError** – *member* was not found in *tarchive*

`autometa.common.utilities.`**`write_checksum`**(*infpath: str*, *outfpath: str*) → str

Calculate checksum for *infpath* and write to *outfpath*.

Parameters

- **`infpath`** (*`str`*) – </path/to/input/file>
- **`outfpath`** (*`str`*) – </path/to/output/checksum/file>

Returns

Description of returned object.

Return type

NoneType

Raises

- **`FileNotFoundError`** – Provided *infpath* does not exist
- **`TypeError`** – *infpath* or *outfpath* is not a string

Module contents

autometa.config package

Submodules

autometa.config.databases module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

This file contains the Databases class responsible for configuration handling of Autometa Databases.

`class` `autometa.config.databases.`**`Databases`**(*config=<configparser.ConfigParser object>*, *dryrun=False*, *nproc=2*, *update=False*)

Bases: `object`

Database class containing methods to allow downloading/formatting/updating Autometa database dependencies.

Parameters

- **`config`** (*`config.ConfigParser`*) – Config containing database dependency information. (the default is DEFAULT_CONFIG).
- **`dryrun`** (*`bool`*) – Run through database checking without performing downloads/formatting (the default is False).
- **`nproc`** (*`int`*) – Number of processors to use to perform database formatting. (the default is mp.cpu_count()).
- **`update`** (*`bool`*) – Overwrite existing databases with more up-to-date database files. (the default is False).

`ncbi_dir`

</path/to/databases/markers> SECTIONS : dict keys are *sections* respective to database config sections and values are options within the *sections*.

Type

str </path/to/databases/ncbi> markers_dir : str

`SECTIONS = {'markers': ['bacteria_single_copy', 'bacteria_single_copy_cutoffs', 'archaea_single_copy', 'archaea_single_copy_cutoffs'], 'ncbi': ['nodes', 'names', 'merged', 'delnodes', 'accession2taxid', 'nr']}`

compare_checksums(*section: Optional[str] = None*) → Dict[str, Dict]

Get all invalid database files in *options* from *section* in config. An md5 checksum comparison will be performed between the current and file's remote md5 to ensure file integrity prior to checking the respective file as valid.

Parameters

`section` (str, optional Configure provided *section* Choices include) – 'markers' and 'ncbi'. (default will download/format all database directories)

Returns

dict {section

Return type

{option, option,. . . }, section:{. . . }, . . . }

configure(*section: Optional[str] = None*, *no_checksum: bool = False*) → ConfigParser

Configures Autometa's database dependencies by first checking missing dependencies then comparing checksums to ensure integrity of files.

Download and format databases for all options in each section.

This will only perform the download and formatting if *self.dryrun* is False. This will update out-of-date databases if *self.update* is True.

Parameters

`section` (str, optional Configure provided *section*. Choices include) – 'markers' and 'ncbi'. (default will download/format all database directories) no_checksum : bool, optional Do not perform checksum comparisons (Default is False).

Returns

databases sections.

Return type

configparser.ConfigParser config with updated options in respective

Raises

`ValueError Provided section does not match 'ncbi', or 'markers'.` – ConnectionError A connection issue occurred when connecting to NCBI or GitHub.

download_gtdb_files() → None

download_markers(*options: Iterable*) → None

Download markers database files and amend user config to reflect this.

Parameters

`options` (*iterable*) – iterable containing options in 'markers' section to download.

Returns

Will update provided *options* in *self.config*.

Return type

NoneType

Raises

`ConnectionError` – marker file download failed.

`download_missing`(*section: Optional[str] = None*) → None

Download missing Autometa database dependencies from provided *section*. If no *section* is provided will check all sections.

Parameters
`section` (`str, optional`) – Section to check for missing database files (the default is None). Choices include 'ncbi', and 'markers'.

Returns
Will update provided *section* in *self.config*.

Return type
NoneType

Raises
`ValueError` – Provided *section* does not match 'ncbi' and 'markers'.

`download_ncbi_files`(*options: Iterable*) → None

Download NCBI database files.

Parameters
`options` (`iterable`) – iterable containing options in 'ncbi' section to download.

Returns
Will update provided *options* in *self.config*.

Return type
NoneType

Raises

- `subprocess.CalledProcessError` – NCBI file download with rsync failed.
- `ConnectionError` – NCBI file checksums do not match after file transfer.

`extract_taxdump`() → None

Extract autometa required files from ncbi taxdump.tar.gz archive into ncbi databases directory and update user config with extracted paths.

This only extracts nodes.dmp, names.dmp, merged.dmp and delnodes.dmp from taxdump.tar.gz if the files do not already exist. If *update* was originally supplied as *True* to the Databases instance, then the previous files will be replaced by the new taxdump files.

After successful extraction of the files, a checksum will be written of the archive for future checking.

Returns
Will update *self.config* section *ncbi* with options 'nodes', 'names', 'merged', 'delnodes'

Return type
NoneType

`fix_invalid_checksums`(*section: Optional[str] = None*) → None

Download/Update/Format databases where checksums are out-of-date.

Parameters
`section` (`str, optional`) – Configure provided *section*. Choices include 'markers' and 'ncbi'. (default will download/format all database directories)

Returns
Will update provided *options* in *self.config*.

Return type
NoneType

Raises
`ConnectionError` – Failed to connect to *section* host site.

`format_nr()` → None

Construct a diamond formatted database (nr.dmnd) from *nr* option in *ncbi* section in user config.

NOTE: The checksum 'nr.dmnd.md5' will only be generated if nr.dmnd construction is successful. If the provided *nr* option in *ncbi* is 'nr.gz' the database will be removed after successful database formatting.

Returns
config updated option:'nr' in section:'ncbi'.

Return type
NoneType

`get_missing`(*section: Optional[str] = None*) → Dict[str, Dict]

Get all missing database files in *options* from *sections* in config.

Parameters
`section` (*str, optional*) – Configure provided *section*. Choices include 'markers' and 'ncbi'. (default will download/format all database directories)

Returns
{section:{option, option,…}, section:{…}, …}

Return type
dict

`get_remote_checksum`(*section: str*, *option: str*) → str

Get the checksum from provided *section* respective to *option* in
self.config.

section
[str] section to retrieve for *checksums* section. Choices include: 'ncbi' and 'markers'.

option
[str] *option* in *checksums* section corresponding to the section checksum file.

str
checksum of remote md5 file. e.g. 'hash filename

'

ValueError
'section' must be 'ncbi' or 'markers'

ConnectionError
No internet connection available.

ConnectionError
Failed to connect to host for provided *option*.

`press_hmms()` → None

hmmpress markers hmm database files.

Return type
NoneType

satisfied(*section: Optional[str] = None*, *compare_checksums: bool = False*) → bool

Determines whether all database dependencies are satisfied.

Parameters

- **section** (*str*) – section to retrieve for *checksums* section. Choices include: 'ncbi' and 'markers'.
- **compare_checksums** (*bool, optional*) – Also check if database information is up-to-date with current hosted databases. (default is False).

Returns
True if all database dependencies are satisfied, otherwise False.

Return type
bool

`autometa.config.databases.`**main**()

autometa.config.environ module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Configuration handling for Autometa environment.

`autometa.config.environ.`**bedtools**()

Get bedtools version.

Returns
version of bedtools

Return type
str

`autometa.config.environ.`**bowtie2**()

Get bowtie2 version.

Returns
version of bowtie2

Return type
str

`autometa.config.environ.`**configure**(*config: ConfigParser*) → Tuple[ConfigParser, bool]

Checks executable dependencies necessary to run autometa. Will update *config* with executable dependencies with details: 1. presence/absence of dependency and its location 2. versions

Parameters
config (*configparser.ConfigParser*) – Description of parameter *config*.

Returns
(config, satisfied) config updated with executables details Details: 1. location of executable 2. version of executable config : configparser.ConfigParser satisfied : bool

Return type
2-tuple

`autometa.config.environ.`**`diamond`**`()`

Get diamond version.

Returns
version of diamond

Return type
str

`autometa.config.environ.`**`find_executables`**`()`

Retrieves executable file paths by looking in Autometa dependent executables.

Returns
{executable:</path/to/executable>, . . . }

Return type
dict

`autometa.config.environ.`**`get_versions`**`(`*program: Optional[str] = None*`)` → Union[Dict[str, str], str]

Retrieve versions from all required executable dependencies. If *program* is provided will only return version for *program*.

See: https://stackoverflow.com/a/834451/12671809

Parameters
`program` (*`str, optional`*) – the program to retrieve the version, by default None

Returns
if program is None: dict - {program:version, . . . } if program: str - version

Return type
dict or str

Raises

- **`ValueError`** – *program* is not a string
- **`KeyError`** – *program* is not an executable dependency.

`autometa.config.environ.`**`hmmpress`**`()`

Get hmmpress version.

Returns
version of hmmpress

Return type
str

`autometa.config.environ.`**`hmmscan`**`()`

Get hmmscan version.

Returns
version of hmmscan

Return type
str

`autometa.config.environ.`**`hmmsearch`**`()`

Get hmmsearch version.

Returns
version of hmmsearch

Return type
str

`autometa.config.environ.`**`prodigal`**`()`

Get prodigal version.

Returns
version of prodigal

Return type
str

`autometa.config.environ.`**`samtools`**`()`

Get samtools version.

Returns
version of samtools

Return type
str

autometa.config.utilities module

`autometa.config.utilities.`**`get_config`**`(`*fpath: str*`)` → ConfigParser

Load the config provided at *fpath*.

Parameters
`fpath` (*`str`*) – </path/to/file.config>

Returns
interpolated config object parsed from *fpath*.

Return type
config.ConfigParser

Raises
`FileNotFoundError` – Provided *fpath* does not exist.

`autometa.config.utilities.`**`main`**`()`

`autometa.config.utilities.`**`parse_args`**`(`*fpath: Optional[str] = None*`)` → Namespace

Generate argparse namespace (args) from config file.

Parameters
`fpath` (*`str`*) – </path/to/file.config> (default is DEFAULT_CONFIG in autometa.config)

Returns
namespace typical to parser.parse_args() method from argparse

Return type
argparse.Namespace

Raises
`FileNotFoundError` – provided *fpath* does not exist.

`autometa.config.utilities.`**`put_config`**`(`*config: ConfigParser*, *out: str*`)` → None

Writes *config* to *out* and updates checkpoints checksum.

Parameters

- **config** (*config.ConfigParser*) – configuration containing user provided parameters and files information.
- **out** (*str*) – </path/to/output/file.config>

Return type
NoneType

`autometa.config.utilities.`**`set_home_dir`**`()` → str

Set the *home_dir* in autometa's default configuration (default.config) based on autometa's current location. If the *home_dir* variable is already set, then this will be used as the *home_dir* location.

Returns
</path/to/package/autometa>

Return type
str

`autometa.config.utilities.`**`update_config`**`(`*section: str*, *option: str*, *value: str*, *fpath: str = '/home/docs/checkouts/readthedocs.org/user_builds/autometa/checkouts/main/buil*
→ None

Update *fpath* in *section* for *option* with *value*.

Parameters

- **fpath** (*str*) – </path/to/file.config>
- **section** (*str*) – *section* header to update within *fpath*.
- **option** (*str*) – *option* to update within *section*.
- **value** (*str*) – *value* to update *option*.

Return type
NoneType

Module contents

autometa.datasets package

Module contents

autometa.taxonomy package

Submodules

autometa.taxonomy.database module

`class autometa.taxonomy.database.`**`TaxonomyDatabase`**

Bases: `ABC`

TaxonomyDatabase Abstract Base Class

Abstract methods

1. parse_nodes(self)
2. parse_names(self)

3. parse_merged(self)
4. parse_delnodes(self)
5. convert_accessions_to_taxids(self)

e.g.

class GTDB(TaxonomyDatabase):

def __init__(self, …):
self.nodes = self.parse_nodes() self.names = self.parse_names() self.merged = self.parse_merged() self.delnodes = self.parse_delnodes() …

def parse_nodes(self):
…

def parse_nodes(self):
…

def parse_merged(self):
…

def parse_delnodes(self):
…

def convert_accessions_to_taxids(self, accessions):
…

Available methods (after aforementioned implementations):

1. convert_taxid_dtype
2. name
3. rank
4. parent
5. lineage
6. is_common_ancestor
7. get_lineage_dataframe

Available attributes:

CANONICAL_RANKS UNCLASSIFIED

`CANONICAL_RANKS = ['species', 'genus', 'family', 'order', 'class', 'phylum', 'superkingdom', 'root']`

`UNCLASSIFIED = 'unclassified'`

`abstract` **`convert_accessions_to_taxids`**(*accessions: Dict[str, Set[str]]*) → Tuple[Dict[str, Set[int]], pandas.DataFrame]

Translates subject sequence ids to taxids

Parameters
`accessions` (*`dict`*) – {qseqid: {sseqid, … }, … }

Returns
{qseqid: {taxid, taxid, … }, … }, index=range, cols=[qseqid, sseqid, raw_taxid, …, cleaned_taxid]

Return type
Tuple[Dict[str, Set[int]], pd.DataFrame]

`convert_taxid_dtype`(*taxid: int*) → int

1. Converts the given *taxid* to an integer and checks whether it is positive.

2. Checks whether *taxid* is present in both nodes.dmp and names.dmp. 3a. If (2) is false, will check for corresponding *taxid* in merged.dmp and convert to this then redo (2). 3b. If (2) is true, will return converted taxid. 4. If (3a) is false will look for *taxid* in delnodes.dmp. If present will convert to root (taxid=1)

Parameters
taxid (*int*) – identifier for a taxon in NCBI taxonomy databases - nodes.dmp, names.dmp or merged.dmp

Returns
taxid if the *taxid* is a positive integer and present in either nodes.dmp or names.dmp or taxid recovered from merged.dmp

Return type
int

Raises

- **ValueError** – Provided *taxid* is not a positive integer
- *DatabaseOutOfSyncError* – NCBI databases nodes.dmp, names.dmp and merged.dmp are out of sync with each other

`get_lineage_dataframe`(*taxids: Iterable*, *fillna: bool = True*) → pandas.DataFrame

Given an iterable of taxids generate a pandas DataFrame of their canonical lineages

Parameters

- **taxids** (*iterable*) – *taxids* whose lineage dataframe is being returned
- **fillna** (*bool, optional*) – Whether to fill the empty cells with TaxonomyDatabase.UNCLASSIFIED or not, default True

Returns
index = taxid columns = [superkingdom,phylum,class,order,family,genus,species]

Return type
pd.DataFrame

Example

If you would like to merge the returned DataFrame ('this_df') with another DataFrame ('your_df'). Let's say where you retrieved your taxids:

```
merged_df = pd.merge(
    left=your_df,
    right=this_df,
    how='left',
    left_on=<taxid_column>,
    right_index=True)
```

`is_common_ancestor`(*taxid_A: int*, *taxid_B: int*) → bool

Determines whether the provided taxids have a non-root common ancestor

Parameters

- **taxid_A** (*int*) – taxid in taxonomy database
- **taxid_B** (*int*) – taxid in taxonomy database

Returns
True if taxids share a common ancestor else False

Return type
boolean

lineage(*taxid: int*, *canonical: bool = True*) → List[Dict[str, Union[str, int]]]

Returns the lineage of *taxids* encountered when traversing to root

Parameters

- **taxid** (*int*) – *taxid* in nodes.dmp, whose lineage is being returned
- **canonical** (*bool, optional*) – Lineage includes both canonical and non-canonical ranks when False, and only the canonical ranks when True Canonical ranks include : species, genus , family, order, class, phylum, superkingdom, root

Returns
[{'taxid':taxid, 'rank':rank,'name':name}, …]

Return type
ordered list of dicts

name(*taxid: int*, *rank: Optional[str] = None*) → str

Parses through the names.dmp in search of the given *taxid* and returns its name.

Parameters

- **taxid** (*int*) – *taxid* whose name is being returned
- **rank** (*str, optional*) – If provided, will return *taxid* name at *rank*, by default None Must be a canonical rank, choices: species, genus, family, order, class, phylum, superkingdom Eg. self.name(562, 'genus') would return 'Escherichia', where 562 is the taxid for Escherichia coli

Returns
Name of provided *taxid* if *taxid* is found in names.dmp else TaxonomyDatabase.UNCLASSIFIED

Return type
str

parent(*taxid: int*) → int

Retrieve the parent taxid of provided *taxid*.

Parameters
taxid (*int*) – child taxid to retrieve parent

Returns
Parent taxid if found in nodes otherwise 1

Return type
int

abstract parse_delnodes() → Set[int]

Parses delnodes.dmp such that deleted `taxid`s may be updated with their up-to-date `taxid`s

Returns
{taxid, …}

Return type
set

`abstract parse_merged()` → Dict[int, int]

Parses merged.dmp such that merged `taxid`s may be updated with their up-to-date `taxid`s

Returns
{old_taxid: new_taxid, . . . }

Return type
dict

`abstract parse_names()` → Dict[int, str]

Parses through the names.dmp in search of the given *taxid* and returns its name

Parameters

- **taxid** (*int*) – *taxid* whose name is being returned
- **rank** (*str, optional*) – If provided, will return *taxid* name at *rank*, by default None Must be a canonical rank, choices: species, genus, family, order, class, phylum, superkingdom Eg. self.name(562, ‘genus’) would return ‘Escherichia’, where 562 is the taxid for Escherichia coli

Returns
Name of provided *taxid* if *taxid* is found in names.dmp else TaxonomyDatabase.UNCLASSIFIED

Return type
str

`abstract parse_nodes()` → Dict[int, Dict[str, Union[str, int]]]

Parse the *nodes.dmp* database and set to self.nodes.

Returns
{child_taxid:{‘parent’:parent_taxid,’rank’:rank}, . . . }

Return type
dict

`rank`(*taxid: int*) → str

Return the respective rank of provided *taxid*.

Parameters
taxid (*int*) – *taxid* to retrieve rank from nodes

Returns
rank name if taxid is found in nodes else autoattribute:: autometa.taxonomy.database.TaxonomyDatabase.UNCLASSIFIED

Return type
str

autometa.taxonomy.gtdb module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

File containing definition of the GTDB class and containing functions useful for handling GTDB taxonomy databases

class `autometa.taxonomy.gtdb.`**`GTDB`**(*dbdir: str*, *verbose: bool = True*)

Bases: `TaxonomyDatabase`

Taxonomy utilities for GTDB databases.

`__repr__`()

Operator overloading to return the string representation of the class object

Returns
String representation of the class object

Return type
str

`__str__`()

Operator overloading to return the directory path of the class object

Returns
Directory path of the class object

Return type
str

`convert_accessions_to_taxids`(*accessions: Dict[str, Set[str]]*) → Tuple[Dict[str, Set[int]], pandas.DataFrame]

Translates subject sequence ids to taxids

Parameters
accessions (*dict*) – {qseqid: {sseqid, …}, …}

Returns
{qseqid: {taxid, taxid, …}, …}, index=range, cols=[qseqid, sseqid, raw_taxid, …, cleaned_taxid]

Return type
Tuple[Dict[str, Set[int]], pd.DataFrame]

`parse_delnodes`() → Set[int]

Parse the delnodes.dmp database

Returns
{taxid, …}

Return type
set

`parse_merged`() → Dict[int, int]

Parse the merged.dmp database

Returns
{old_taxid: new_taxid, …}

Return type
dict

parse_names() → Dict[int, str]

Parses through names.dmp database and loads taxids with scientific names

Returns

{taxid:name, …}

Return type

dict

parse_nodes() → Dict[int, str]

Parse the *nodes.dmp* database. Note: This is performed when a new GTDB class instance is constructed

Returns

{child_taxid:{'parent':parent_taxid,'rank':rank}, …}

Return type

dict

search_genome_accessions(*accessions: set*) → Dict[str, int]

Search taxid.map file

Parameters

accessions (`set`) – Set of subject sequence ids retrieved from diamond blastp search (sseqids)

Returns

Dictionary containing sseqids converted to taxids

Return type

Dict[str, int]

verify_databases()

Verify if the required databases are present.

Raises

FileNotFoundError – One or more of the required database were not found.

autometa.taxonomy.gtdb.**create_gtdb_db**(*reps_faa: str*, *dbdir: str*) → str

Generate a combined faa file to create the GTDB-t database.

Parameters

- **reps_faa** (`str`) – Directory having faa file of all representative genomes. Can be tarballed.
- **dbdir** (`str`) – Path to output directory.

Returns

Path to combined faa file. This can be used to make a diamond database.

Return type

str

autometa.taxonomy.gtdb.**main**()

autometa.taxonomy.lca module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

This script contains the LCA class containing methods to determine the Lowest Common Ancestor given a tab-delimited BLAST table, fasta file, or iterable of SeqRecords.

Note: LCA will assume the BLAST results table is in output format 6.

`class autometa.taxonomy.lca.`**`LCA`**(*taxonomy_db:* TaxonomyDatabase, *verbose: bool = False*, *cache: str = ''*)

Bases: `object`

LCA class containing methods to retrieve the Lowest Common Ancestor.

LCAs may be computed given taxids, a fasta or BLAST results.

Parameters

- **dbdir** (*str*) – Path to directory containing files: nodes.dmp, names.dmp, merged.dmp, prot.accession2taxid.gz
- **outdir** (*str*) – Output directory path to to serialize intermediate files to disk for later lookup
- **verbose** (*bool, optional*) – Add verbosity to logging stream (the default is False).

`disable`

Opposite of verbose. Used to disable *tqdm* module.

Type
bool

`tour_fp`

</path/to/serialized/file/eulerian/tour.pkl.gz>

Type
str

`tour`

Eulerian tour containing branches and leaves information from tree traversal.

Type
list

`level_fp`

</path/to/serialized/file/level.pkl.gz>

Type
str

`level`

Lengths from root corresponding to *tour* during tree traversal.

Type
list

`occurrence_fp`

</path/to/serialized/file/level.pkl.gz>

Type
str

`occurrence`

Contains first occurrence of each taxid while traversing tree (index in *tour*). e.g. {taxid:index, taxid: index, . . . }

Type
dict

`sparse_fp`

</path/to/serialized/file/sparse.pkl.gz>

Type
str

`sparse`

Precomputed LCA values corresponding to *tour*,`level` and *occurrence*.

Type
numpy.ndarray

`lca_prepared`

Whether LCA internals have been computed (e.g. *tour*,`level`,`occurrence`,`sparse`).

Type
bool

`blast2lca`(*blast: str*, *out: str*, *sseqid_to_taxid_output: str = ''*, *lca_reduction_log: str = ''*, *force: bool = False*) → str

Determine lowest common ancestor of provided amino-acid ORFs.

Parameters

- **`blast`** (*`str`*) – </path/to/diamond/outfmt6/blastp.tsv>.
- **`out`** (*`str`*) – </path/to/output/lca.tsv>.
- **`sseqid_to_taxid_output`** (*`str`*) – Path to write qseqids' sseqids and their taxid designations from NCBI databases
- **`force`** (*`bool, optional`*) – Force overwrite of existing *out*.

Returns
out </path/to/output/lca.tsv>.

Return type
str

`lca`(*node1*, *node2*)

Performs Range Minimum Query between 2 taxids.

Parameters

- **`node1`** (*`int`*) – taxid
- **`node2`** (*`int`*) – taxid

Returns
LCA taxid

Return type
int

Raises
`ValueError` – Provided taxid is not in the nodes.dmp tree.

parse(*lca_fpath: str*, *orfs_fpath: Optional[str] = None*) → Dict[str, Dict[str, Dict[int, int]]]

Retrieve and construct contig dictionary from provided *lca_fpath*.

Parameters

- **lca_fpath** (*str*) – </path/to/lcas.tsv> tab-delimited ordered columns: qseqid, name, rank, lca_taxid
- **orfs_fpath** (*str, optional (required if using prodigal version <2.6)*) – </path/to/prodigal/called/orfs.fasta> Note: These ORFs should correspond to the ORFs provided in the BLAST table.

Returns

{contig:{rank:{taxid:counts, …}, rank:{…}, …}, …}

Return type

dict

Raises

- **FileNotFoundError** – *lca_fpath* does not exist.
- **FileNotFoundError** – *orfs_fpath* does not exist.
- **ValueError** – If prodigal version is under 2.6, *orfs_fpath* is a required input.

prepare_lca()

Prepare LCA internal data structures for `lca()`.

e.g. self.tour, self.level, self.occurrence, self.sparse are all ready.

Returns

Prepares all LCA internals and if successful sets *self.lca_prepared* to True.

Return type

NoneType

prepare_tree()

Performs Eulerian tour of nodes.dmp taxids and constructs three data structures:

1. tour : list of branches and leaves.
2. level: list of distances from the root.
3. occurrence: dict of occurences of the taxid respective to the root.

Notes

For more information on why we construct these three data structures see references below:

- **`Geeksforgeeks: Find LCA in Binary Tree using RMQ https://www.geeksforgeeks.org/find-lca-in-binary-tree-using-rmq/`_**
- **`Topcoder: Another easy solution in <O(N logN, O(logN)> https://www.topcoder.com/community/competitive-programming/tutorials/range-minimum-query-and-lowest-common-ancestor/#Another%20easy%20solution%20in%20O(N%20logN,%20O(logN)`_**

Returns

sets internals to be used for LCA lookup

Return type

NoneType

preprocess_minimums()

Preprocesses all possible LCAs.

This constructs a sparse table to be used for LCA/Range Minimum Query using the *self.level* array associated with its respective eulerian *self.tour*. For more information on these data structures see `prepare_tree()`.

Sparse table size:

- n = number of elements in level list
- rows range = (0 to n)
- columns range = (0 to logn)

Returns
sets *self.sparse* internal to be used for LCA lookup.

Return type
NoneType

read_sseqid_to_taxid_table(*sseqid_to_taxid_filepath: str*) → Dict[str, Set[int]]

Retrieve each qseqid's set of taxids from *sseqid_to_taxid_filepath* for reduction by LCA

Parameters
sseqid_to_taxid_filepath (*str*) – Path to sseqid to taxid table with columns: qseqid, sseqid, raw_taxid, merged_taxid, clean_taxid

Returns
Dictionary keyed by qseqid containing sets of respective *clean* taxid

Return type
Dict[str, Set[int]]

reduce_taxids_to_lcas(*taxids: Dict[str, Set[int]]*) → Tuple[Dict[str, int], pandas.DataFrame]

Retrieves the lowest common ancestor for each set of taxids in of the taxids

Parameters
taxids (*dict*) – {qseqid: {taxid, . . . }, qseqid: {taxid, . . . }, . . . }

Returns
{qseqid: lca, qseqid: lca, . . . }, pd.DataFrame(index=range, cols=[qseqid, taxids])

Return type
Tuple[Dict[str, int], pd.DataFrame]

write_lcas(*lcas: Dict[str, int]*, *out: str*) → str

Write *lcas* to tab-delimited file: *out*.

Ordered columns are:

- qseqid : query seqid
- name : LCA name
- rank : LCA rank
- lca : LCA taxid

Parameters

- **lcas** (*dict*) – {qseqid:lca_taxid, qseqid:lca_taxid, . . . }

- **out** (*str*) – </path/to/output/file.tsv>

Returns
out

Return type
str

`autometa.taxonomy.lca.main()`

autometa.taxonomy.majority_vote module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

This script contains the modified majority vote algorithm used in Autometa version 1.0

`autometa.taxonomy.majority_vote.`**`is_consistent_with_other_orfs`**(*taxid: int*, *rank: str*, *rank_counts: Dict[str, Dict]*, *taxa_db: TaxonomyDatabase*) → bool

Determines whether the majority of proteins in a contig, with rank equal to or above the given rank, are common ancestors of the taxid.

If the majority are, this function returns True, otherwise it returns False.

Parameters

- **taxid** (*int*) – *taxid* to search against other taxids at *rank* in *rank_counts*.
- **rank** (*str*) – Canonical rank to search in *rank_counts*. Choices: species, genus, family, order, class, phylum, superkingdom.
- **rank_counts** (*dict*) – LCA canonical rank counts retrieved from ORFs respective to a contig. e.g. {canonical_rank: {taxid: num_hits, … }, … }
- **ncbi** (*NCBI instance*) – Instance or subclass of NCBI from autometa.taxonomy.ncbi.

Returns
If the majority of ORFs in a contig are equal or above given rank then return True, otherwise return False.

Return type
boolean

`autometa.taxonomy.majority_vote.`**`lowest_majority`**(*rank_counts: Dict[str, Dict]*, *taxa_db: TaxonomyDatabase*) → int

Determine the lowest majority given *rank_counts* by first attempting to get a taxid that leads in counts with the highest specificity in terms of canonical rank.

Parameters

- **rank_counts** (*dict*) – {canonical_rank:{taxid:num_hits, … }, rank2: { … }, … }
- **taxa_db** (*TaxonomyDatabase instance*) – NCBI or GTDB subclass object of autometa.taxonomy.database.TaxonomyDatabase

Returns
Taxid above the lowest majority threshold.

Return type
int

autometa.taxonomy.majority_vote.**main**()

autometa.taxonomy.majority_vote.**majority_vote**(*lca_fpath: str*, *out: str*, *taxa_db:* TaxonomyDatabase, *verbose: bool = False*, *orfs: Optional[str] = None*) → str

Wrapper for modified majority voting algorithm from Autometa 1.0

Parameters

- **lca_fpath** (*str*) – Path to lowest common ancestor assignments table.
- **out** (*str*) – Path to write assigned taxids.
- **taxa_db** (TaxonomyDatabase) – An instance of TaxonomyDatabase
- **verbose** (*bool, optional*) – Increase verbosity of logging stream
- **orfs** (*str, optional*) – Path to prodigal called orfs corresponding to LCA table computed from BLAST output
- **force** (*bool, optional*) – Whether to overwrite existing LCA results.

Returns
Path to assigned taxids table.

Return type
str

autometa.taxonomy.majority_vote.**rank_taxids**(*ctg_lcas: Dict[str, Dict[str, Dict[int, int]]]*, *taxa_db:* TaxonomyDatabase, *verbose: bool = False*) → Dict[str, int]

Votes for taxids based on modified majority vote system where if a majority does not exist, the lowest majority is voted.

Parameters

- **ctg_lcas** (*dict*) – {ctg1:{canonical_rank:{taxid:num_hits,…},…}, ctg2:{…},…}
- **taxa_db** (TaxonomyDatabase) – instance of NCBI or GTDB subclass of autometa.taxonomy.database.TaxonomyDatabase
- **verbose** (*bool*) – Description of parameter *verbose* (the default is False).

Returns
{contig:voted_taxid, contig:voted_taxid, …}

Return type
dict

autometa.taxonomy.majority_vote.**write_votes**(*results: Dict[str, int]*, *out: str*) → str

Writes voting *results* to provided *outfpath*.

Parameters

- **results** (*dict*) – {contig:voted_taxid, contig:voted_taxid, …}
- **out** (*str*) – </path/to/results.tsv>.

Returns
</path/to/results.tsv>

Return type
str

Raises
`FileExistsError` – Voting results file already exists

autometa.taxonomy.ncbi module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

File containing definition of the NCBI class and containing functions useful for handling NCBI taxonomy databases

`class autometa.taxonomy.ncbi.`**`NCBI`**`(`*dbdir*, *verbose=False*`)`

Bases: *TaxonomyDatabase*

Taxonomy utilities for NCBI databases.

`__repr__``()`

Operator overloading to return the string representation of the class object

Returns
String representation of the class object

Return type
str

`__str__``()`

Operator overloading to return the directory path of the class object

Returns
Directory path of the class object

Return type
str

`convert_accessions_to_taxids``(`*accessions: set*`)` → Tuple[Dict[str, Set[int]], pandas.DataFrame]

Translates subject sequence ids to taxids

Parameters
accessions (*dict*) – {qseqid: {sseqid, …}, …}

Returns
{qseqid: {taxid, taxid, …}, …}, index=range, cols=[qseqid, sseqid, raw_taxid, …, cleaned_taxid]

Return type
Tuple[Dict[str, Set[int]], pd.DataFrame]

`convert_taxid_dtype``(`*taxid: int*`)` → int

1. Converts the given *taxid* to an integer and checks whether it is positive.

2. Checks whether *taxid* is present in both nodes.dmp and names.dmp. 3a. If (2) is false, will check for corresponding *taxid* in merged.dmp and convert to this then redo (2). 3b. If (2) is true, will return converted taxid. 4. If (3a) is false will look for *taxid* in delnodes.dmp. If present will convert to root (taxid=1)

Parameters
taxid (*int*) – identifier for a taxon in NCBI taxonomy databases - nodes.dmp, names.dmp or merged.dmp

Returns
taxid if the *taxid* is a positive integer and present in either nodes.dmp or names.dmp or taxid recovered from merged.dmp

Return type
int

Raises

- `ValueError` – Provided *taxid* is not a positive integer
- *`DatabaseOutOfSyncError`* – NCBI databases nodes.dmp, names.dmp and merged.dmp are out of sync with each other

`parse_delnodes()` → Set[int]

Parse the delnodes.dmp database Note: This is performed when a new NCBI class instance is constructed

Returns
{taxid, …}

Return type
set

`parse_merged()` → Dict[int, int]

Parse the merged.dmp database Note: This is performed when a new NCBI class instance is constructed

Returns
{old_taxid: new_taxid, …}

Return type
dict

`parse_names()` → Dict[int, str]

Parses through names.dmp database and loads taxids with scientific names

Returns
{taxid:name, …}

Return type
dict

`parse_nodes()` → Dict[int, str]

Parse the *nodes.dmp* database to be used later by `autometa.taxonomy.ncbi.NCBI.parent()`, `autometa.taxonomy.ncbi.NCBI.rank()` Note: This is performed when a new NCBI class instance is constructed

Returns
{child_taxid:{'parent':parent_taxid,'rank':rank}, …}

Return type
dict

`search_prot_accessions`(*accessions: set*, *sseqids_to_taxids: Optional[Dict[str, int]] = None*, *db: str = 'live'*) → Dict[str, int]

Search prot.accession2taxid.gz and dead_prot.accession2taxid.gz

Parameters

- **accessions** (*`set`*) – Set of subject sequence ids retrieved from diamond blastp search (sseqids)
- **sseqids_to_taxids** (*`Dict[str, int], optional`*) – Dictionary containing sseqids converted to taxids
- **db** (*`str, optional`*) – selection of one of the prot accession to taxid databases from NCBI. Choices are live, dead, full

  - live: prot.accession2taxid.gz
  - full: prot.accession2taxid.FULL.gz
  - dead: dead_prot.accession2taxid.gz

Returns
Dictionary containing sseqids converted to taxids

Return type
Dict[str, int]

autometa.taxonomy.vote module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Script to split metagenome assembly by kingdoms given the input votes. The lineages of the provided voted taxids will also be added and written to taxonomy.tsv

`autometa.taxonomy.vote.`**`add_ranks`**(*df: pandas.DataFrame*, *taxa_db:* TaxonomyDatabase) → pandas.DataFrame

Add canonical ranks to *df* and write to *out*

Parameters

- **df** (*pd.DataFrame*) – index="contig", column="taxid"
- **taxa_db** (TaxonomyDatabase) – NCBI or GTDB TaxonomyDatabase instance.

Returns
index="contig", columns=["taxid", *canonical_ranks]

Return type
pd.DataFrame

`autometa.taxonomy.vote.`**`assign`**(*out: str*, *method: str = 'majority_vote'*, *assembly: Optional[str] = None*, *prot_orfs: Optional[str] = None*, *nucl_orfs: Optional[str] = None*, *blast: Optional[str] = None*, *lca_fpath: Optional[str] = None*, *dbdir: str = './autometa/databases/ncbi'*, *dbtype: Literal['ncbi', 'gtdb'] = 'ncbi'*, *force: bool = False*, *verbose: bool = False*, *parallel: bool = True*, *cpus: int = 0*) → pandas.DataFrame

Assign taxonomy using *method* and write to *out*.

Parameters

- **out** (*str*) – Path to write taxonomy table of votes
- **method** (*str, optional*) – Method to assign contig taxonomy, by default "majority_vote". choices include "majority_vote", …
- **assembly** (*str, optional*) – Path to assembly fasta file (nucleotide), by default None
- **prot_orfs** (*str, optional*) – Path to amino-acid ORFs called from *assembly*, by default None
- **nucl_orfs** (*str, optional*) – Path to nucleotide ORFs called from *assembly*, by default None
- **blast** (*str, optional*) – Path to blastp table, by default None
- **lca_fpath** (*str, optional*) – Path to output of LCA analysis, by default None

- **`dbdir`** (*`str, optional`*) – Path to NCBI databases directory, by default NCBI_DIR
- **`dbtype`** (*`str, optional`*) – Type of Taxonomy database to use, by default ncbi
- **`force`** (*`bool, optional`*) – Overwrite existing annotations, by default False
- **`verbose`** (*`bool, optional`*) – Increase verbosity, by default False
- **`parallel`** (*`bool, optional`*) – Whether to perform annotations using multiprocessing and GNU parallel, by default True
- **`cpus`** (*`int, optional`*) – Number of cpus to use if *parallel* is True, by default will try to use all available.

Returns
index=”contig”, columns=[“taxid”]

Return type
pd.DataFrame

Raises

- **`NotImplementedError`** – Provided *method* has not yet been implemented.
- **`ValueError`** – Assembly file is required if no other annotations are provided.

`autometa.taxonomy.vote.`**`get`**`(`*filepath_or_dataframe: Union[str, pandas.DataFrame]*, *kingdom: str*, *taxa_db: TaxonomyDatabase*`)` → pandas.DataFrame

Retrieve specific *kingdom* voted taxa for *assembly* from *filepath*

Parameters

- **`filepath`** (*`str`*) – Path to tab-delimited taxonomy table. cols=[‘contig’,’taxid’, *canonical_ranks]
- **`kingdom`** (*`str`*) – rank to retrieve from superkingdom column in taxonomy table.
- **`ncbi`** (*`str or autometa.taxonomy.NCBI instance, optional`*) – Path to NCBI database directory or NCBI instance, by default NCBI_DIR. This is necessary only if *filepath* does not already contain columns of canonical ranks.

Returns
DataFrame of contigs pertaining to retrieved *kingdom*.

Return type
pd.DataFrame

Raises

- **`FileNotFoundError`** – Provided *filepath* does not exists or is empty.
- ***`TableFormatError`*** – Provided *filepath* does not contain the ‘superkingdom’ column.
- **`KeyError`** – *kingdom* is absent in provided taxonomy table.

`autometa.taxonomy.vote.`**`main`**`()`

`autometa.taxonomy.vote.`**`write_ranks`**`(`*taxonomy: pandas.DataFrame*, *assembly: str*, *outdir: str*, *rank: str = 'superkingdom'*, *prefix: Optional[str] = None*`)` → List[str]

Write fastas split by *rank*

Parameters

- **`taxonomy`** (*`pd.DataFrame`*) – dataframe containing canonical ranks of contigs assigned from :func:autometa.taxonomy.vote.assign(…)

- **assembly** (*str*) – Path to assembly fasta file
- **outdir** (*str*) – Path to output directory to write fasta files
- **rank** (*str, optional*) – canonical rank column in taxonomy table to split by, by default “superkingdom”
- **prefix** (*str, optional*) – Prefix each of the paths written with *prefix* string.

Returns
[rank_name_fpath, . . .]

Return type
list

Raises
`KeyError` – *rank* not in taxonomy columns

Module contents

autometa.validation package

Submodules

autometa.validation.assess_metagenome_deconvolution module

autometa.validation.benchmark module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

Autometa taxon-profiling, clustering and binning-classification evaluation benchmarking.

Script to benchmark Autometa taxon-profiling, clustering and binning-classification results using clustering and classification evaluation metrics.

`class autometa.validation.benchmark.`**`Labels`**`(`*true*, *pred*`)`

Bases: `tuple`

`pred`

Alias for field number 1

`true`

Alias for field number 0

`class autometa.validation.benchmark.`**`Targets`**`(`*true*, *pred*, *target_names*`)`

Bases: `tuple`

`pred`

Alias for field number 1

`target_names`

Alias for field number 2

`true`

Alias for field number 0

`automata.validation.benchmark.`**`compute_classification_metrics`**(*labels: namedtuple*) → dict

Retrieve classification report from using scikit-learn's metrics.classification_report method.

Parameters
`labels` (*`namedtuple`*) – Labels namedtuple containing 'true' and 'pred' fields

Returns
Computed classification metrics corresponding to *labels*

Return type
dict

`autometa.validation.benchmark.`**`compute_clustering_metrics`**(*labels: NamedTuple*, *average_method: str*) → Dict[str, float]

Calculate various clustering performance metrics listed below.

Note: Some of these clustering performance evaluation metrics adjust for chance. This is discussed in more detail in the scikit-learn user guide. - Adjustment for chance in clustering performance evaluation

This analysis suggests the most robust and reliable metrics to use as the number of clusters increases are adjusted rand index and adjusted mutual info score.

- adjusted mutual info score
- geometric normalized mutual info score
- adjusted rand index
- homogeneity
- completeness
- V-measure
- fowlkes-mallows score

Parameters

- **`labels`** (`Labels`) – Labels NamedTuple with *Labels.pred* (predictions) and *Labels.true* (reference) namespaces
- **`average_method`** (*`str`*) – Normalizer to select for normalized mutual information score clustering metric. choices: min, geometric, arithmetic, max

Returns
computed clustering evaluation metrics keyed by their metric

Return type
Dict[str, float]

Raises

- **`ValueError`** – The input arguments are not the correct type (pd.DataFrame or str)
- **`ValueError`** – The provided reference community and predictions do not match!

`autometa.validation.benchmark.`**`evaluate_binning_classification`**(*predictions: Iterable*, *reference: str*) → pandas.DataFrame `autometa.validation.benchmark.`**`evaluate_classification`**(*predictions: Iterable*, *reference: str*, *ncbi: Union[str, NCBI]*, *keep_averages=['weighted avg', 'samples avg']*) → Tuple[pandas.DataFrame, List[Dict[str, str]]]

Evaluate classification *predictions* against provided *reference*

Parameters

- **`predictions`** (*`Iterable`*) – Paths to taxonomic predictions (tab-delimited files of contig and taxid columns)
- **`reference`** (*`str`*) – Path to ground truths (tab-delimited file containing at least contig and taxid columns)
- **`ncbi`** (*`Union[str, NCBI]`*) – Path to NCBI databases directory or instance of autometa NCBI class
- **`keep_averages`** (*`list, optional`*) – averages to keep from classification report, by default [“weighted avg”, “samples avg”]

Returns

Metrics

Return type

Tuple[pd.DataFrame, List[dict]]

`autometa.validation.benchmark.`**`evaluate_clustering`**(*predictions: Iterable*, *reference: str*, *average_method: str*) → pandas.DataFrame

Evaluate clustering performance of *predictions* against *reference*

Parameters

- **`predictions`** (*`Iterable`*) – Paths to binning predictions. Paths should be tab-delimited files with ‘cluster’ and ‘contig’ columns.
- **`reference`** (*`str`*) – Path to ground truth reference genome assignments. Should be tab-delimited file with ‘contig’ and ‘reference_genome’ columns.
- **`average_method`** (*`str`*) – Normalizer to select for normalized mutual information score clustering metric. choices: min, geometric, arithmetic, max

Returns

Dataframe of clustering metrics indexed by ‘dataset’ computed as each predictions basename

Return type

pd.DataFrame

`autometa.validation.benchmark.`**`get_categorical_labels`**(*predictions: str*, *reference: Union[str, pandas.DataFrame]*) → NamedTuple

Retrieve categorical labels from *predictions* and *reference*

Parameters

- **`predictions`** (*`str`*) – Path to tab-delimited file containing contig clustering predictions with columns ‘contig’ and ‘cluster’.
- **`reference`** (*`Union[str, pd.DataFrame]`*) – Path to tab-delimited file containing ground truth reference genome assignments with columns ‘contig’ and ‘reference_genome’.

Returns

Labels namedtuple with ‘true’ and ‘pred’ fields containing respective dataframes of categorical values

Return type
NamedTuple

Raises

- **`ValueError`** – Provided *reference* is not a pd.DataFrame or path to ground-truth reference genome assignments file.
- **`ValueError`** – The provided *reference* community contigs do not match the *predictions* contigs

`autometa.validation.benchmark.`**`get_target_labels`**(*prediction: str*, *reference: Union[str, pandas.DataFrame]*, *ncbi: Union[str, NCBI]*) → namedtuple

Retrieve taxid lineage as target labels from merge of *reference* and *prediction*.

Note: The exact label value matters for these metrics as we are looking at the available target labels for classification (not clustering)

Parameters

- **`prediction`** (*`str`*) – Path to contig taxid predictions
- **`reference`** (*`Union[str, pd.DataFrame]`*) – Path to ground truth contig taxids
- **`ncbi`** (*`Union[str, NCBI]`*) – Path to NCBI databases directory or instance of autometa NCBI class.

Returns
Targets namedtuple with fields 'true', 'pred' and 'target_names'

Return type
namedtuple

Raises

- **`ValueError`** – Provided reference is not a pd.DataFrame or path to reference assignments file.
- **`ValueError`** – The provided reference community and predictions do not match

`autometa.validation.benchmark.`**`main`**()

`autometa.validation.benchmark.`**`write_reports`**(*reports: Iterable[Dict]*, *outdir: str*, *ncbi: NCBI*) → None

Write taxid multi-label classification reports in *reports*

Parameters

- **`reports`** (*`Iterable[Dict]`*) – List of classification report dicts from each classification benchmarking evaluation
- **`outdir`** (*`str`*) – Directory path to write reports
- **`ncbi`** (`NCBI`) – autometa.taxonomy.ncbi.NCBI instance for taxid name and rank look-up.

Return type
NoneType

autometa.validation.build_protein_marker_aln module

autometa.validation.calculate_f1_scores module

autometa.validation.cami module

Autometa module to format Autometa results to bioboxes data format (compatible with CAMI tools)

Reformats Autometa binning results such that they may be submitted to CAMI for benchmarking assessment

`autometa.validation.cami.`**`format_genome_binning`**(*df: pandas.DataFrame*, *sample_id: str*, *version: str = '0.9.0'*) → str

Format autometa genome binning results to be compatible with specified BioBox *version*.

Parameters

- **`genome_binning`** (`Union[str,pd.DataFrame]`) – Path to (to-be-formatted) genome_binning results or *pd.DataFrame(index_col=range(0, n_rows), columns=[contig, taxid, …])*
- **`sample_id`** (`str`) – Sample identifier, not the generating user or program name. It MUST match the regular expression *[A-Za-z0-9._]+*.
- **`version`** (`str, optional`) – Biobox version to format results, by default *"0.9"*

Returns
formatted results ready to be written to a file path

Return type
str

Raises

- **`NotImplementedError`** – Specified *version* is not implemented
- **`TypeError`** – *genome_binning* must be a path to a taxon results file or pandas
- **`ValueError`** – *genome_binning* does not contain the required columns

`autometa.validation.cami.`**`format_profiling`**(*df: pandas.DataFrame*, *sample_id: str*, *version: str*) → str

`autometa.validation.cami.`**`format_taxon_binning`**(*df: pandas.DataFrame*, *sample_id: str*, *version: str = '0.9.0'*) → str

Format autometa taxon binning results to be compatible with specified BioBox *version*.

Parameters

- **`taxon_binning`** (`Union[str,pd.DataFrame]`) – Path to (to-be-formatted) taxon_binning results or *pd.DataFrame(index_col=range(0, n_rows), columns=[contig, taxid, …])*
- **`sample_id`** (`str`) – Sample identifier, not the generating user or program name. It MUST match the regular expression *[A-Za-z0-9._]+*.
- **`version`** (`str, optional`) – Biobox version to format results, by default *"0.9"*

Returns
formatted results ready to be written to a file path

Return type
str

Raises

- **`NotImplementedError`** – Specified *version* is not implemented
- **`TypeError`** – *taxon_binning* must be a path to a taxon results file or pandas
- **`ValueError`** – *taxon_binning* does not contain the required columns

`autometa.validation.cami.`**`get_biobox_format`**(*predictions: Union[str, pandas.DataFrame]*, *sample_id: str*, *results_type: Literal['profiling', 'genome_binning', 'taxon_binning']*, *version: str*) → str

`autometa.validation.cami.`**`main`**()

autometa.validation.cluster_process module

Processes metagenome assembled genomes from autometa binning results

`autometa.validation.cluster_process.`**`assess_assembly`**(*seq_record_list*)

`autometa.validation.cluster_process.`**`main`**(*args*)

`autometa.validation.cluster_process.`**`run_command`**(*command_string*, *stdout_path=None*)

autometa.validation.cluster_process_docker module

autometa.validation.cluster_taxonomy module

autometa.validation.compile_reference_training_table module

autometa.validation.confidence_vs_accuracy module

autometa.validation.datasets module

# License: GNU Affero General Public License v3 or later # A copy of GNU AGPL v3 should have been included in this software package in LICENSE.txt.

pulling data from google drive dataset with simulated or synthetic communities

`autometa.validation.datasets.`**`download`**(*community_type: str*, *community_sizes: list*, *file_names: list*, *dir_path: str*) → None

Downloads the files specified in a dictionary.

Parameters

- **`community_type`** (`str`) – specifies the type of dataset that the user would like to download from
- **`community_sizes`** (`list`) – specifies the size of dataset that the user would like to download
- **`file_names`** (`list`) – specifies the file(s) that the user would like to download
- **`dir_path`** (`str`) – output path where the user wants to download the file(s)

Returns

download is completed through gdown

Return type

None

autometa.validation.datasets.**main**()

autometa.validation.download_random_bacterial_genomes module

autometa.validation.length_vs_accuracy module

autometa.validation.make_simulated_metagenome module

autometa.validation.make_simulated_metagenome_control_fasta module

autometa.validation.reference_genome_from_quast module

autometa.validation.show_clusters module

autometa.validation.summarize_f1_stats module

autometa.validation.tabulate_bins module

autometa.validation.tabulate_metaquast_alignments module

autometa.validation.vizualize_assembly_graph_by_bin module

Module contents

Module contents

1.13 Legacy Autometa

To run Autometa version 1, you will need to download the latest Autometa version 1 release or navigate to the latest Autometa version 1 commit on the GitHub repository.

The latest Autometa version 1 release may be found here: https://github.com/KwanLab/Autometa/releases

You may find *any* of the Autometa releases by selecting the release and looking under the release tag on the left.

1.14 License

GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling.

In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source" link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see <https://www.gnu.org/licenses/>.

1.15 Contact

If you like Autometa, visit our lab website to find out more about our research.
For suggestions, queries or appreciation feel free to contact Dr. Jason Kwan at jason.kwan@wisc.edu

1.16 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

autometa, 141
autometa.binning, 87
autometa.binning.large_data_mode, 73
autometa.binning.large_data_mode_loginfo, 75
autometa.binning.recursive_dbscan, 77
autometa.binning.summary, 81
autometa.binning.utilities, 83
autometa.common, 111
autometa.common.coverage, 95
autometa.common.exceptions, 97
autometa.common.external, 95
autometa.common.external.bedtools, 87
autometa.common.external.bowtie, 88
autometa.common.external.diamond, 89
autometa.common.external.hmmscan, 91
autometa.common.external.hmmsearch, 93
autometa.common.external.prodigal, 93
autometa.common.external.samtools, 95
autometa.common.kmers, 98
autometa.common.markers, 102
autometa.common.metagenome, 104
autometa.common.utilities, 107
autometa.config, 118
autometa.config.databases, 111
autometa.config.environ, 115
autometa.config.utilities, 117
autometa.datasets, 118
autometa.taxonomy, 135
autometa.taxonomy.database, 118
autometa.taxonomy.gtdb, 123
autometa.taxonomy.lca, 125
autometa.taxonomy.majority_vote, 129
autometa.taxonomy.ncbi, 131
autometa.taxonomy.vote, 133
autometa.validation, 141
autometa.validation.benchmark, 135
autometa.validation.cami, 139
autometa.validation.cluster_process, 140
autometa.validation.datasets, 140

INDEX

Symbols

`__repr__()` (*autometa.taxonomy.gtdb.GTDB method*), 123
`__repr__()` (*autometa.taxonomy.ncbi.NCBI method*), 131
`__str__()` (*autometa.common.exceptions.DatabaseOutOfSyncError method*), 97
`__str__()` (*autometa.taxonomy.gtdb.GTDB method*), 123
`__str__()` (*autometa.taxonomy.ncbi.NCBI method*), 131

A

`add_clustering_runtime_summary_info()` (*in module autometa.binning.large_data_mode_loginfo*), 75
`add_embedding_runtime_summary_info()` (*in module autometa.binning.large_data_mode_loginfo*), 76
`add_metrics()` (*in module autometa.binning.utilities*), 83
`add_ranks()` (*in module autometa.taxonomy.vote*), 133
`aggregate_orfs()` (*in module autometa.common.external.prodigal*), 93
`align()` (*in module autometa.common.external.bowtie*), 88
`annotate_parallel()` (*in module autometa.common.external.hmmscan*), 91
`annotate_parallel()` (*in module autometa.common.external.prodigal*), 93
`annotate_sequential()` (*in module autometa.common.external.hmmscan*), 91
`annotate_sequential()` (*in module autometa.common.external.prodigal*), 93
`apply_binning_metrics_filter()` (*in module autometa.binning.utilities*), 84
`assess_assembly()` (*in module autometa.validation.cluster_process*), 140
`assign()` (*in module autometa.taxonomy.vote*), 133
`autometa`
    `module`, 141
`autometa.binning`
    `module`, 87
`autometa.binning.large_data_mode`
    `module`, 73
`autometa.binning.large_data_mode_loginfo`
    `module`, 75
`autometa.binning.recursive_dbscan`
    `module`, 77
`autometa.binning.summary`
    `module`, 81
`autometa.binning.utilities`
    `module`, 83
`autometa.common`
    `module`, 111
`autometa.common.coverage`
    `module`, 95
`autometa.common.exceptions`
    `module`, 97
`autometa.common.external`
    `module`, 95
`autometa.common.external.bedtools`
    `module`, 87
`autometa.common.external.bowtie`
    `module`, 88
`autometa.common.external.diamond`
    `module`, 89
`autometa.common.external.hmmscan`
    `module`, 91
`autometa.common.external.hmmsearch`
    `module`, 93
`autometa.common.external.prodigal`
    `module`, 93
`autometa.common.external.samtools`
    `module`, 95
`autometa.common.kmers`
    `module`, 98
`autometa.common.markers`
    `module`, 102
`autometa.common.metagenome`
    `module`, 104
`autometa.common.utilities`
    `module`, 107

autometa.config
    module, 118
autometa.config.databases
    module, 111
autometa.config.environ
    module, 115
autometa.config.utilities
    module, 117
autometa.datasets
    module, 118
autometa.taxonomy
    module, 135
autometa.taxonomy.database
    module, 118
autometa.taxonomy.gtdb
    module, 123
autometa.taxonomy.lca
    module, 125
autometa.taxonomy.majority_vote
    module, 129
autometa.taxonomy.ncbi
    module, 131
autometa.taxonomy.vote
    module, 133
autometa.validation
    module, 141
autometa.validation.benchmark
    module, 135
autometa.validation.cami
    module, 139
autometa.validation.cluster_process
    module, 140
autometa.validation.datasets
    module, 140
autometa_clr() (*in module autometa.common.kmers*), 98
AutometaError, 97

B

bedtools() (*in module autometa.config.environ*), 115
BinningError, 97
blast() (*in module autometa.common.external.diamond*), 89
blast2lca() (*autometa.taxonomy.lca.LCA method*), 126
bowtie2() (*in module autometa.config.environ*), 115
build() (*in module autometa.common.external.bowtie*), 88

C

calc_checksum() (*in module autometa.common.utilities*), 107
CANONICAL_RANKS (*autometa.taxonomy.database.TaxonomyDatabase attribute*), 119
checkpoint() (*in module autometa.binning.large_data_mode*), 73
ChecksumMismatchError, 97
cluster_by_taxon_partitioning() (*in module autometa.binning.large_data_mode*), 73
compare_checksums() (*autometa.config.databases.Databases method*), 112
compute_classification_metrics() (*in module autometa.validation.benchmark*), 135
compute_clustering_metrics() (*in module autometa.validation.benchmark*), 136
configure() (*autometa.config.databases.Databases method*), 112
configure() (*in module autometa.config.environ*), 115
contigs_from_headers() (*in module autometa.common.external.prodigal*), 93
convert_accessions_to_taxids() (*autometa.taxonomy.database.TaxonomyDatabase method*), 119
convert_accessions_to_taxids() (*autometa.taxonomy.gtdb.GTDB method*), 123
convert_accessions_to_taxids() (*autometa.taxonomy.ncbi.NCBI method*), 131
convert_taxid_dtype() (*autometa.taxonomy.database.TaxonomyDatabase method*), 120
convert_taxid_dtype() (*autometa.taxonomy.ncbi.NCBI method*), 131
count() (*in module autometa.common.kmers*), 98
create_gtdb_db() (*in module autometa.taxonomy.gtdb*), 124

D

DatabaseOutOfSyncError, 97
Databases (*class in autometa.config.databases*), 111
describe() (*autometa.common.metagenome.Metagenome method*), 105
diamond() (*in module autometa.config.environ*), 115
disable (*autometa.taxonomy.lca.LCA attribute*), 125
download() (*in module autometa.validation.datasets*), 140
download_gtdb_files() (*autometa.config.databases.Databases method*), 112
download_markers() (*autometa.config.databases.Databases method*), 112
download_missing() (*autometa.config.databases.Databases method*), 112

download_ncbi_files() (*autometa.config.databases.Databases method*), 113

E

embed() (*in module autometa.common.kmers*), 99
evaluate_binning_classification() (*in module autometa.validation.benchmark*), 136
evaluate_classification() (*in module autometa.validation.benchmark*), 136
evaluate_clustering() (*in module autometa.validation.benchmark*), 137
ExternalToolError, 97
extract_taxdump() (*autometa.config.databases.Databases method*), 113

F

file_length() (*in module autometa.common.utilities*), 107
filter_domtblout() (*in module autometa.common.external.hmmsearch*), 93
filter_taxonomy() (*in module autometa.binning.utilities*), 84
filter_tblout_markers() (*in module autometa.common.external.hmmscan*), 91
find_executables() (*in module autometa.config.environ*), 116
fix_invalid_checksums() (*autometa.config.databases.Databases method*), 113
format_genome_binning() (*in module autometa.validation.cami*), 139
format_nr() (*autometa.config.databases.Databases method*), 114
format_profiling() (*in module autometa.validation.cami*), 139
format_taxon_binning() (*in module autometa.validation.cami*), 139
format_total_times() (*in module autometa.binning.large_data_mode_loginfo*), 76
fragmentation_metric() (*autometa.common.metagenome.Metagenome method*), 105
fragmentation_metric() (*in module autometa.binning.summary*), 81
from_spades_names() (*in module autometa.common.coverage*), 95

G

gc_content() (*autometa.common.metagenome.Metagenome method*), 105
genomecov() (*in module autometa.common.external.bedtools*), 87
get() (*in module autometa.common.coverage*), 96
get() (*in module autometa.common.markers*), 102
get() (*in module autometa.taxonomy.vote*), 134
get_agg_stats() (*in module autometa.binning.summary*), 82
get_biobox_format() (*in module autometa.validation.cami*), 140
get_categorical_labels() (*in module autometa.validation.benchmark*), 137
get_checkpoint_info() (*in module autometa.binning.large_data_mode*), 74
get_clusters() (*in module autometa.binning.recursive_dbscan*), 77
get_config() (*in module autometa.config.utilities*), 117
get_kmer_embedding() (*in module autometa.binning.large_data_mode*), 75
get_lineage_dataframe() (*autometa.taxonomy.database.TaxonomyDatabase method*), 120
get_loginfo() (*in module autometa.binning.large_data_mode_loginfo*), 76
get_metabin_stats() (*in module autometa.binning.summary*), 82
get_metabin_taxonomies() (*in module autometa.binning.summary*), 82
get_missing() (*autometa.config.databases.Databases method*), 114
get_remote_checksum() (*autometa.config.databases.Databases method*), 114
get_target_labels() (*in module autometa.validation.benchmark*), 138
get_versions() (*in module autometa.config.environ*), 116
GTDB (*class in autometa.taxonomy.gtdb*), 123
gunzip() (*in module autometa.common.utilities*), 107

H

hmmpress() (*in module autometa.common.external.hmmscan*), 91
hmmpress() (*in module autometa.config.environ*), 116
hmmscan() (*in module autometa.config.environ*), 116
hmmsearch() (*in module autometa.config.environ*), 116

I

init_kmers() (*in module autometa.common.kmers*), 101
internet_is_connected() (*in module autometa.common.utilities*), 108
is_common_ancestor() (*autometa.taxonomy.database.TaxonomyDatabase

method), 120
is_consistent_with_other_orfs() (*in module autometa.taxonomy.majority_vote*), 129
is_gz_file() (*in module autometa.common.utilities*), 108

L

Labels (*class in autometa.validation.benchmark*), 135
largest_seq (*autometa.common.metagenome.Metagenome attribute*), 104
largest_seq (*autometa.common.metagenome.Metagenome property*), 105
LCA (*class in autometa.taxonomy.lca*), 125
lca() (*autometa.taxonomy.lca.LCA method*), 126
lca_prepared (*autometa.taxonomy.lca.LCA attribute*), 126
length_filter() (*autometa.common.metagenome.Metagenome method*), 105
length_weighted_gc (*autometa.common.metagenome.Metagenome attribute*), 104
length_weighted_gc (*autometa.common.metagenome.Metagenome property*), 106
level (*autometa.taxonomy.lca.LCA attribute*), 125
level_fp (*autometa.taxonomy.lca.LCA attribute*), 125
lineage() (*autometa.taxonomy.database.TaxonomyDatabase method*), 121
load() (*in module autometa.common.kmers*), 101
load() (*in module autometa.common.markers*), 103
lowest_majority() (*in module autometa.taxonomy.majority_vote*), 129

M

main() (*in module autometa.binning.large_data_mode*), 75
main() (*in module autometa.binning.large_data_mode_loginfo*), 77
main() (*in module autometa.binning.recursive_dbscan*), 78
main() (*in module autometa.binning.summary*), 83
main() (*in module autometa.common.coverage*), 97
main() (*in module autometa.common.external.bedtools*), 87
main() (*in module autometa.common.external.bowtie*), 88
main() (*in module autometa.common.external.hmmscan*), 91
main() (*in module autometa.common.external.hmmsearch*), 93
main() (*in module autometa.common.external.prodigal*), 94
main() (*in module autometa.common.external.samtools*), 95
main() (*in module autometa.common.kmers*), 101
main() (*in module autometa.common.markers*), 104
main() (*in module autometa.common.metagenome*), 107
main() (*in module autometa.config.databases*), 115
main() (*in module autometa.config.utilities*), 117
main() (*in module autometa.taxonomy.gtdb*), 124
main() (*in module autometa.taxonomy.lca*), 129
main() (*in module autometa.taxonomy.majority_vote*), 129
main() (*in module autometa.taxonomy.vote*), 134
main() (*in module autometa.validation.benchmark*), 138
main() (*in module autometa.validation.cami*), 140
main() (*in module autometa.validation.cluster_process*), 140
main() (*in module autometa.validation.datasets*), 140
majority_vote() (*in module autometa.taxonomy.majority_vote*), 130
make_pickle() (*in module autometa.common.utilities*), 108
makedatabase() (*in module autometa.common.external.diamond*), 90
Metagenome (*class in autometa.common.metagenome*), 104
module
  autometa, 141
  autometa.binning, 87
  autometa.binning.large_data_mode, 73
  autometa.binning.large_data_mode_loginfo, 75
  autometa.binning.recursive_dbscan, 77
  autometa.binning.summary, 81
  autometa.binning.utilities, 83
  autometa.common, 111
  autometa.common.coverage, 95
  autometa.common.exceptions, 97
  autometa.common.external, 95
  autometa.common.external.bedtools, 87
  autometa.common.external.bowtie, 88
  autometa.common.external.diamond, 89
  autometa.common.external.hmmscan, 91
  autometa.common.external.hmmsearch, 93
  autometa.common.external.prodigal, 93
  autometa.common.external.samtools, 95
  autometa.common.kmers, 98
  autometa.common.markers, 102
  autometa.common.metagenome, 104
  autometa.common.utilities, 107
  autometa.config, 118
  autometa.config.databases, 111
  autometa.config.environ, 115
  autometa.config.utilities, 117
  autometa.datasets, 118

    autometa.taxonomy, 135
    autometa.taxonomy.database, 118
    autometa.taxonomy.gtdb, 123
    autometa.taxonomy.lca, 125
    autometa.taxonomy.majority_vote, 129
    autometa.taxonomy.ncbi, 131
    autometa.taxonomy.vote, 133
    autometa.validation, 141
    autometa.validation.benchmark, 135
    autometa.validation.cami, 139
    autometa.validation.cluster_process, 140
    autometa.validation.datasets, 140
mp_counter() (*in module autometa.common.kmers*), 101

N

name() (*autometa.taxonomy.database.TaxonomyDatabase method*), 121
NCBI (*class in autometa.taxonomy.ncbi*), 131
ncbi_dir (*autometa.config.databases.Databases attribute*), 111
ncbi_is_connected() (*in module autometa.common.utilities*), 108
normalize() (*in module autometa.common.kmers*), 101
nseqs (*autometa.common.metagenome.Metagenome attribute*), 104
nseqs (*autometa.common.metagenome.Metagenome property*), 106

O

occurrence (*autometa.taxonomy.lca.LCA attribute*), 125
occurrence_fp (*autometa.taxonomy.lca.LCA attribute*), 125
orf_records_from_contigs() (*in module autometa.common.external.prodigal*), 94

P

parent() (*autometa.taxonomy.database.TaxonomyDatabase method*), 121
parse() (*autometa.taxonomy.lca.LCA method*), 126
parse() (*in module autometa.common.external.bedtools*), 87
parse() (*in module autometa.common.external.diamond*), 90
parse_args() (*in module autometa.config.utilities*), 117
parse_delnodes() (*autometa.taxonomy.database.TaxonomyDatabase method*), 121
parse_delnodes() (*autometa.taxonomy.gtdb.GTDB method*), 123
parse_delnodes() (*autometa.taxonomy.ncbi.NCBI method*), 132
parse_merged() (*autometa.taxonomy.database.TaxonomyDatabase method*), 122
parse_merged() (*autometa.taxonomy.gtdb.GTDB method*), 123
parse_merged() (*autometa.taxonomy.ncbi.NCBI method*), 132
parse_names() (*autometa.taxonomy.database.TaxonomyDatabase method*), 122
parse_names() (*autometa.taxonomy.gtdb.GTDB method*), 123
parse_names() (*autometa.taxonomy.ncbi.NCBI method*), 132
parse_nodes() (*autometa.taxonomy.database.TaxonomyDatabase method*), 122
parse_nodes() (*autometa.taxonomy.gtdb.GTDB method*), 124
parse_nodes() (*autometa.taxonomy.ncbi.NCBI method*), 132
pred (*autometa.validation.benchmark.Labels attribute*), 135
pred (*autometa.validation.benchmark.Targets attribute*), 135
prepare_lca() (*autometa.taxonomy.lca.LCA method*), 127
prepare_tree() (*autometa.taxonomy.lca.LCA method*), 127
preprocess_minimums() (*autometa.taxonomy.lca.LCA method*), 127
press_hmms() (*autometa.config.databases.Databases method*), 114
prodigal() (*in module autometa.config.environ*), 117
put_config() (*in module autometa.config.utilities*), 117

R

rank() (*autometa.taxonomy.database.TaxonomyDatabase method*), 122
rank_taxids() (*in module autometa.taxonomy.majority_vote*), 130
read_annotations() (*in module autometa.binning.utilities*), 85
read_checksum() (*in module autometa.common.utilities*), 109
read_domtblout() (*in module autometa.common.external.hmmscan*), 91
read_sseqid_to_taxid_table() (*autometa.taxonomy.lca.LCA method*), 128
read_tblout() (*in module autometa.common.external.hmmscan*), 92
record_counter() (*in module autometa.common.kmers*), 102
recursive_dbscan() (*in module autometa.binning.recursive_dbscan*), 78
recursive_hdbscan() (*in module autometa.binning.recursive_dbscan*), 78
reduce_taxids_to_lcas() (*autometa.taxonomy.lca.LCA method*), 128

`reindex_bin_names()` (*in module autometa.binning.utilities*), 85
`run()` (*in module autometa.common.external.bowtie*), 89
`run()` (*in module autometa.common.external.hmmscan*), 92
`run()` (*in module autometa.common.external.prodigal*), 94
`run_command()` (*in module autometa.validation.cluster_process*), 140
`run_dbscan()` (*in module autometa.binning.recursive_dbscan*), 79
`run_hdbscan()` (*in module autometa.binning.recursive_dbscan*), 80

S

`samtools()` (*in module autometa.config.environ*), 117
`satisfied()` (*autometa.config.databases.Databases method*), 114
`search_genome_accessions()` (*autometa.taxonomy.gtdb.GTDB method*), 124
`search_prot_accessions()` (*autometa.taxonomy.ncbi.NCBI method*), 132
`SECTIONS` (*autometa.config.databases.Databases attribute*), 111
`seq_counter()` (*in module autometa.common.kmers*), 102
`seqrecords` (*autometa.common.metagenome.Metagenome attribute*), 104
`seqrecords` (*autometa.common.metagenome.Metagenome property*), 106
`sequences` (*autometa.common.metagenome.Metagenome attribute*), 104
`sequences` (*autometa.common.metagenome.Metagenome property*), 106
`set_home_dir()` (*in module autometa.config.utilities*), 118
`size` (*autometa.common.metagenome.Metagenome attribute*), 104
`size` (*autometa.common.metagenome.Metagenome property*), 106
`sort()` (*in module autometa.common.external.samtools*), 95
`sparse` (*autometa.taxonomy.lca.LCA attribute*), 126
`sparse_fp` (*autometa.taxonomy.lca.LCA attribute*), 126

T

`TableFormatError`, 97
`tarchive_results()` (*in module autometa.common.utilities*), 109
`target_names` (*autometa.validation.benchmark.Targets attribute*), 135
`Targets` (*class in autometa.validation.benchmark*), 135
`taxon_guided_binning()` (*in module autometa.binning.recursive_dbscan*), 80
`TaxonomyDatabase` (*class in autometa.taxonomy.database*), 118
`timeit()` (*in module autometa.common.utilities*), 109
`tour` (*autometa.taxonomy.lca.LCA attribute*), 125
`tour_fp` (*autometa.taxonomy.lca.LCA attribute*), 125
`true` (*autometa.validation.benchmark.Labels attribute*), 135
`true` (*autometa.validation.benchmark.Targets attribute*), 135

U

`UNCLASSIFIED` (*autometa.taxonomy.database.TaxonomyDatabase attribute*), 119
`unpickle()` (*in module autometa.common.utilities*), 110
`untar()` (*in module autometa.common.utilities*), 110
`update_config()` (*in module autometa.config.utilities*), 118

V

`verify_databases()` (*autometa.taxonomy.gtdb.GTDB method*), 124

W

`write_checksum()` (*in module autometa.common.utilities*), 110
`write_cluster_records()` (*in module autometa.binning.summary*), 83
`write_lcas()` (*autometa.taxonomy.lca.LCA method*), 128
`write_ranks()` (*in module autometa.taxonomy.vote*), 134
`write_reports()` (*in module autometa.validation.benchmark*), 138
`write_results()` (*in module autometa.binning.utilities*), 86
`write_votes()` (*in module autometa.taxonomy.majority_vote*), 130

Z

`zero_pad_bin_names()` (*in module autometa.binning.utilities*), 86