Autometa

Release 2.0

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

Apr 15, 2022

CONTENTS

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

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, 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 Conda*.

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/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

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 conda (I'd recommend it, its almost foolproof!), you'll need to first install Miniconda 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 Miniconda installer to your home directory.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O $HOME/
→Miniconda3-latest-Linux-x86_64.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/Miniconda3-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:

```
cd $HOME/miniconda3/bin/
./conda init
```

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

```
source ~/.bashrc
```

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

```
conda 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 conda 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 conda environment. To activate the environment you'll need to run the following command:

```
conda 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".

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 conda environment is activated. You can activate your environment by running:

```
conda 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".
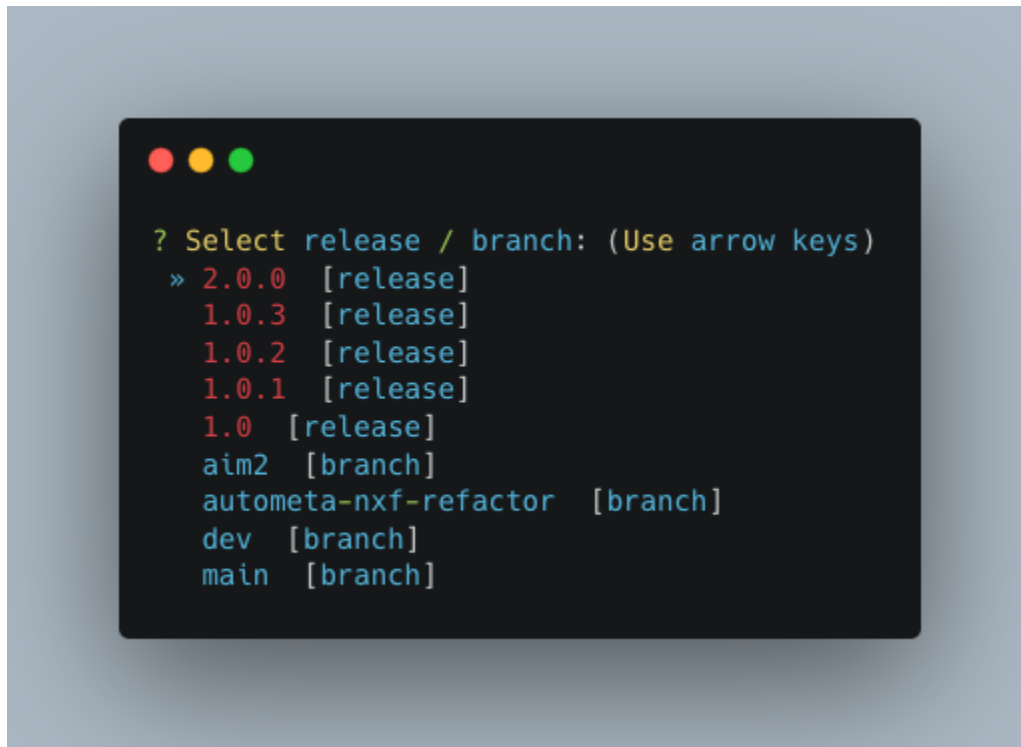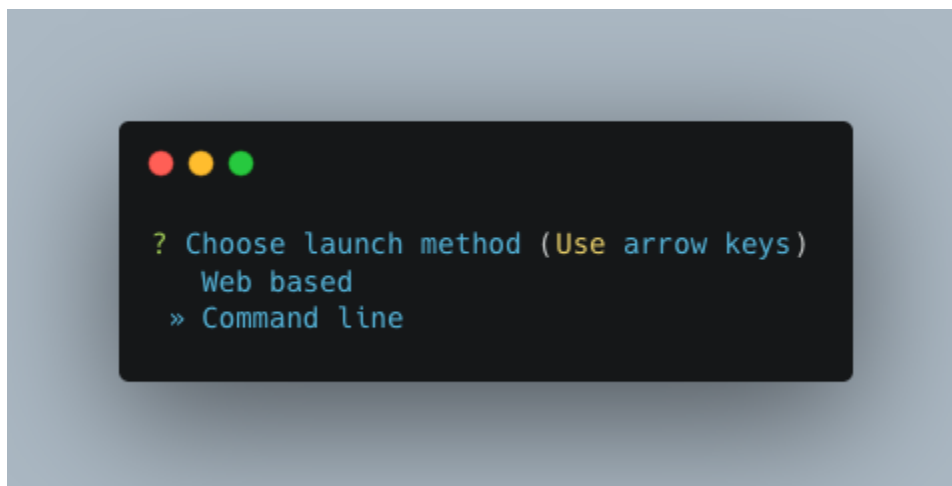
Choose nf-core interface

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.

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]
```
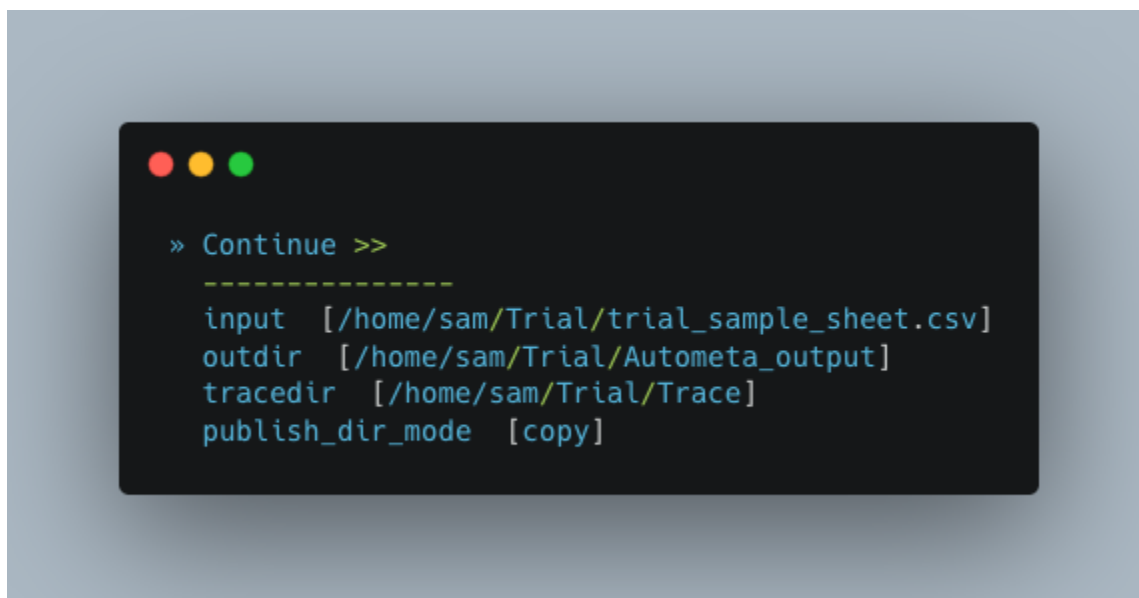
Input and Output

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).

Additional Autometa options

Here you have a choice to make:
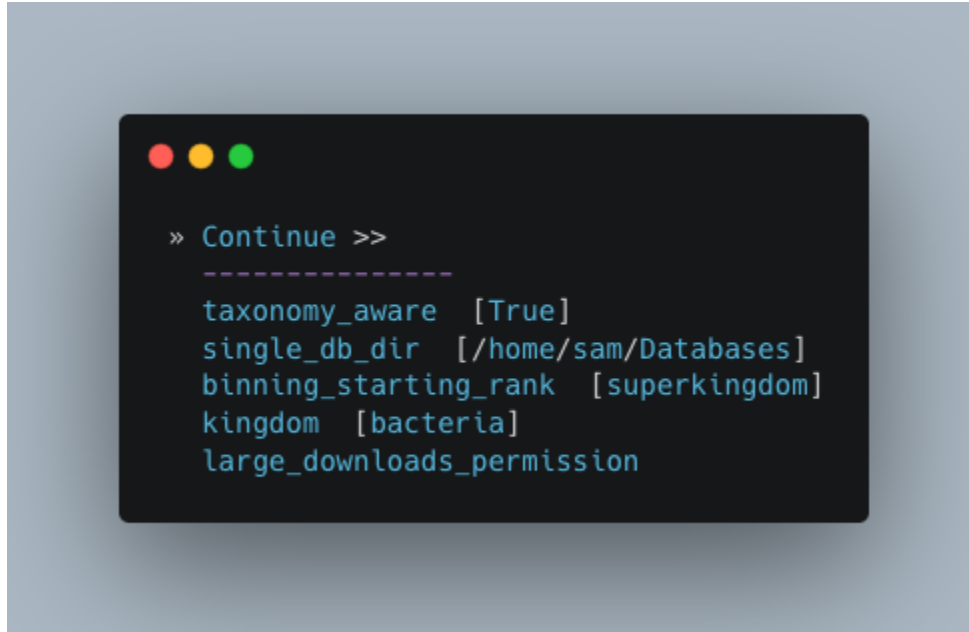
- By enabling taxonomy aware mode, Autometa will attempt to use taxonomic data to make your bins more accurate.

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.

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

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.

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 Conda environment to install Nextflow and nf-core tools and then running the Autometa pipeline.

Installing Nextflow and nf-core tools with Conda

If you have not previously installed/used Conda, you can get it using the Miniconda installer appropriate to your system, here: https://docs.conda.io/en/latest/miniconda.html

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

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

If you receive the message…

```
CondaValueError: prefix already exists:
```

…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.

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

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

```
conda 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.

- `conda`: Enables running all processes using conda

- `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 (`conda 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)

```
        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 Conda, you can get it using the Miniconda installer appropriate to your system, here: https://docs.conda.io/en/latest/miniconda.html

After installing conda, running the following command will create a minimal Conda environment named "autometa".

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

If you receive the message. . .

```
CondaValueError: prefix already exists:
```

. . . 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.

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

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

```
conda activate 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`

```
conda 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. `conda 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. `conda 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 conda 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 Step by Step Tutorial

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

Before running anything make sure you have activated the conda environment using `conda 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.tsv	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-output`	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-write	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.fna	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-dimensions`	Number of dimensions to reduce to PCA feature space after normalization and prior to embedding (default: 50)	Optional
`--embedding-output`	Path to embedded k-mer table	Required
`--embedding-method`	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 \
    --clustering-method dbscan \
    --completeness 20 \
    --purity 90 \
    --cov-stddev-limit 25 \
    --gc-stddev-limit 5 \
    --taxonomy $HOME/tutorial/78mbp_metagenome.taxonomy.tsv \
    --output-binning $HOME/tutorial/78mbp_metagenome.binning.tsv \
    --output-main $HOME/tutorial/78mbp_metagenome.main.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-binning`	Path to write Autometa binning results	Required
`--output-main`	Path to write Autometa main table	Required
`--clustering-method`	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-limit`	coverage standard deviation limit to retain cluster (default 25)	Optional
`--gc-stddev-limit`	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
`--domain`	Kingdom to consider. Choices bacteria (default) and archaea	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_stddev	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 (DB-SCAN) 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-binning`	Path to write Autometa unclustered recruitment table	Y
`--taxonomy`	Path to taxonomy table	N
`--output-features`	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

1.5.1 Markers

Autometa comes packaged with the necessary markers files. 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

If you are running Autometa for the first time you will need to download the NCBI 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 download Autometa (See *Installation*).

```
# 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

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.

1.6 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')

# 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"
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 `conda` then `gdown` should already be installed. If not, you can install it using `conda 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

```python
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.

```
# 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))
$ 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 conda and docker. For installation using conda, we suggest downloading miniconda.

> **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 miniconda

2. Create a new environment with autometa installed: `conda create -c bioconda -n autometa autometa`

3. Activate autometa environment `conda activate autometa`

1.8.2 Install from source (using make)

Download and install miniconda. 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
```

(continues on next page)

```
# Navigate into the cloned repository
cd Autometa

# create autometa conda environment
make create_environment

# activate autometa conda environment
conda 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 miniconda. Now run the following commands:

```
# Construct the autometa environment from autometa-env.yml
conda env create --file=https://raw.githubusercontent.com/KwanLab/Autometa/main/autometa-
↪env.yml

# Activate environment
conda activate autometa

# Install the autometa code base from source
python setup.py install
```

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 conda environment
conda 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 *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.

```python
from autometa.common.external import samtools

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

# 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

```python
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. . .

```python
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,
    method="densmap"
)
```

1.10 Usage

1.10.1 common

external

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`
```

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
```

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>

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
```

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.
```

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)
```

diamond.py

```

```

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
```

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
```

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

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)
```

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
                          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)
```

1.10.2 binning

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}
                        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)
```

utilities.py

```
```

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)
  --ncbi dirpath       Path to user NCBI databases directory (Required for
                       retrieving metabin taxonomies) (default: None)
  --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)
```

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
                        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)
```

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
                          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)
```

1.10.3 taxonomy

ncbi.py

```
```

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 NCBI databases directory. (default: NCBI_DIR)
  --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 NCBI databases directory. (default: NCBI_DIR)
  --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)
```

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      Directory to output fasta files of split canonical
                        ranks and taxonomy.tsv. (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)
  --ncbi dirpath           Path to NCBI databases directory. (default: NCBI_DIR)
```

1.10.4 config

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}
                           config section to update
  --option OPTION          option in `--section` to update
  --value VALUE            Value to update `--option`
```

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. (default: False)
  --update-markers         Update out-of-date markers databases. (default: False)
  --update-ncbi            Update out-of-date ncbi 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)

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

1.11 Contributing Guidelines

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 conda environment
conda activate autometa
# Install dependencies
conda 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 conda environment
conda 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
```

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:

```python
@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:

```python
@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.binning.large_data_mode_loginfo module

autometa.binning.recursive_dbscan module

autometa.binning.summary module

autometa.binning.unclustered_recruitment module

autometa.binning.utilities module

Module contents

autometa.common package

Subpackages

autometa.common.external package

Submodules

automata.common.external.bedtools module

automata.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.

automata.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

automata.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

automata.common.external.bowtie.**main**()

automata.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

autometa.common.external.hmmscan module

autometa.common.external.hmmsearch module

autometa.common.external.prodigal module

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

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: *autometa.common.exceptions.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 automata.common.exceptions.**ChecksumMismatchError**
    Bases: *automata.common.exceptions.AutometaError*

ChecksumMismatchError exception class

Exception called when checksums do not match.

exception automata.common.exceptions.**DatabaseOutOfSyncError**(*value*)
    Bases: *automata.common.exceptions.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 automata.common.exceptions.**ExternalToolError**(*cmd*, *err*)
    Bases: *automata.common.exceptions.AutometaError*

Raised when samtools sort is not executed properly.

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

exception automata.common.exceptions.**TableFormatError**
    Bases: *automata.common.exceptions.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.

automata.common.kmers module

automata.common.markers module

automata.common.metagenome module

automata.common.utilities module

Module contents

automata.config package

Submodules

automata.config.databases module

automata.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.ConfigParser*`)` → Tuple[configparser.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.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*) → argparse.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.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>

>> **Returns**

> **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/2.0.1/auto`
`→ 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*.

> **Returns**

> **Return type** NoneType

Module contents

autometa.datasets package

Module contents

autometa.taxonomy package

Submodules

autometa.taxonomy.lca module

autometa.taxonomy.majority_vote module

autometa.taxonomy.ncbi module

autometa.taxonomy.vote module

Module contents

autometa.validation package

Submodules

autometa.validation.assess_metagenome_deconvolution module

autometa.validation.benchmark module

autometa.validation.build_protein_marker_aln module

autometa.validation.calculate_f1_scores module

autometa.validation.cluster_process module

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

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 the commit associated with 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

Symbols

P

R

S

T

U