SSR_VibraProfiler

SSR_VibraProfiler is a tool for constructing DNA fingerprint databases from whole-genome SSR features and predicting the varieties of unknown individuals.

The workflow of SSR_VibraProfiler is organized into the following four core modules:

- Recognize model construction.
- Recognize model evaluation.
- Individual variety identification.
- Cross-validation.

Installation

The recommended installation method for all users is Docker, as it includes all necessary code, dependencies, and test datasets pre-installed.

Alternatively, installation via Conda or from the source code is also supported.

> Note**:** If you choose to run any tools directly from the source code (rather than using Docker or Conda), please ensure that they are correctly added to your system's environment variables.

Platform requirements

Regardless of the installation method, SSR_VibraProfiler must be run in a **linux command-line environment**. For users without access to a Linux platform, it is recommended to use Docker Desktop as an alternative.

Installation by docker (recommend)

Requirement:

- docker/docker-desktop

Installation：

Shell

```
docker pull oldcat931/ssr_vibraprofiler:1.01
```

Installation from Conda or source code

Requirement:

- Minia
- MISA (only misa.pl is needed)
- Python 3.9.1
- Conda (the version of Conda we used is 23.9.0)

Make sure they are all installed and added to the environment variables.

Optional requirement:

- Arial font package – If not installed, a fallback font will be used automatically. This will not affect the execution of the tool.

Installation Minia

Shell

```
git clone --recursive https://github.com/GATB/minia.git
cd minia
sh INSTALL
```

Please ensure that the Minia working directory is added to your environment variables.

For example, if Minia is located in `/project1/bin/`, users can add it with the following command:

Shell

```
echo 'export PATH=/project1/bin/minia/build/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Minia can also be installed via **Conda**, in which case there is no need to manually set the environment variable.

Shell

```
conda install minia
```

To check whether Minia is working correctly, type the following command in the terminal:

Shell

```
minia --v
```

Users should see an output similar to:

Shell

```
Minia version 3.2.5
Using gatb-core version 1.4.2
OS: Linux-6.8.0-52-generic
compiler: /usr/bin/cc  (11.4.0)
```

Installation MISA

Shell

```
wget "https://webblast.ipk-gatersleben.de/misa/misa_sourcecode_25082020.zip"
unzip misa_sourcecode_25082020.zip && rm misa.ini
```

Please add the directory containing `misa.pl` to environment variables.

For example, if `misa.pl` is located in `/project1/bin/`, run:

Shell

```
echo 'export PATH=/project1/bin/:$PATH' >> ~/.bashrc
source ~/.bashrc
```

To check whether MISA is installed and working properly, enter the following command in your terminal:

Shell

```
misa.pl
```

You should see output similar to the following:

Plain Text

```
_______________________________________________________________________________
DESCRIPTION: Tool for the identification and localization of
             (I)  perfect microsatellites as well as
             (II) compound microsatellites (two individual microsatellites,
                  disrupted by a certain number of bases)
SYNTAX:   misa.pl <FASTA file>
   <FASTAfile>    Single file in FASTA format containing the sequence(s).
   In order to specify the search criteria, an additional file containing
   the microsatellite search parameters is required named "misa.ini", which
   has the following structure:
     (a) Following a text string beginning with 'def', pairs of numbers are
         expected, whereas the first number defines the unit size and the
         second number the lower threshold of repeats for that specific unit.
     (b) Following a text string beginning with 'int' a single number defines
         the maximal number of bases between two adjacent microsatellites in
         order to specify the compound microsatellite type.
     (c) Following a text string beginning with 'GFF' a single string is expected
         either 'true' or 'false' to indicate with optional GFF(v3) output
         should be provided.
   Example:
     definition(unit_size,min_repeats):          1-10 2-6 3-5 4-5 5-5 6-5
     interruptions(max_difference_for_2_SSRs):   100
EXAMPLE: misa.pl seqs.fasta
_______________________________________________________________________________
```

Installation SSR_VibraProfiler

Installation from Conda:

Shell

```
conda install -c oldcat931 ssr_vibraprofiler
```

Installation from Source Code

Shell

```
git clone https://github.com/Olcat35412/SSR_VibraProfiler
cd SSR_VibraProfiler
conda env create -f SSR_VibraProfiler.yml
conda activate ssr_vibraprofiler_env
chmod +x *
```

Please manually add the script directory to **environment variables**.

For example, if the path is in `/project1/bin/SSR_VibraProfiler/`, run:

Shell

```
echo 'export PATH=/project1/bin/SSR_VibraProfiler:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Verify Installation

To check if the scripts are working properly, try running the following command:

Shell

```
SSR_VibraProfiler_cross_validation.py
```

Users should receive an output similar to:

Shell

```
usage: SSR_VibraProfiler_cross_validation.py [-h] -index INDEX_PATH -p
p PP -o OUTPUT_PATH -log LOG_PATH
A Variety Recognition Model Based on Whole Genome SSR Digital Features
optional arguments:
  -h, --help            show this help message and exit
  -index INDEX_PATH, --index_path INDEX_PATH
  -pp PP
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
  -log LOG_PATH, --log_path LOG_PATH
```

Users may repeat this check with the other three scripts as well.

Usage and operation example in a lightweight dataset of *Rhododendron*

Our original *Rhododendron* dataset has a total size of 800 GB. To improve usability and facilitate testing, we performed downsampling to create a lightweight version (~1 GB).

This lightweight dataset is **pre-packaged in the Docker image**.

For users who are not using Docker, it can be downloaded manually with the following command:

Shell

```
wget "https://download.scidb.cn/download?fileId=200a381497fccebcfbebe85
8fda429f2&path=/V2/lightweight_dataset.tar.gz&username=jiangchen1234@16
3.com&fileName=lightweight_dataset.tar.gz" -O lightweight_dataset.tar.g
z
tar -xzf lightweight_dataset.tar.gz
```

The original ***Rhododendron*** dataset can also be obtained using the following command:

Shell

```
wget "https://china.scidb.cn/download?fileId=2779cd477eecc01f550144809f6b9e9a&username=jiangchen1234@163.com&traceId=a62cbd41-3e6d-482f-a101-6c2e9d1b8967"
```

Get Started with Docker

To start SSR_VibraProfiler using Docker, run the following command:

Shell

```
docker run -itd --name SSR_VibraProfiler oldcat931/ssr_vibraprofiler:1.0
docker exec -it SSR_VibraProfiler /bin/bash
```

When using your own dataset, it is recommended to use the `-v` option to reduce disk usage and specify an output directory. For example:

Shell

```
docker run -itd -v data_path:/project1/data_volume -v output_path:/project1/output --name SSR_VibraProfiler oldcat931/ssr_vibraprofiler:1.0
docker exec -it SSR_VibraProfiler /bin/bash
```

Please replace `data_path` with the path to the folder where your dataset is stored, and `output_path` with the path to your desired output directory.

Script Usage

Recognize model construction——SSR_VibraProfiler_model_build.py

This script integrates three key steps. Users can can optionally execute the first two steps by themselves if desired:

1) Assembly of sequencing data using Minia.

2) Extraction of SSR information using MISA.

3) Screening of SSR information and model construction.

Usage and Options:

Shell

```
SSR_VibraProfiler_model_build.py -s -e -pp -index -o -misap -miniap -mi
niac
```

The following parameters are **essential**:

- `-s`: Specifies the start stage.
- `-e`: Specifies the end stage.
- `-pp`: Indicates the polymorphism (as described in the main text of the paper.)
- `-o`: Designates the output path for storing the final identification model and SSR lists.
- `-index`: Refers to the index file. The index file in this tool is used to specify the input individual files and assign each individual its "corresponding label" (variety). Here is an example:

Shell

```
1    /project1/data/lightweight_dataset/W-1-01_li
1    /project1/data/lightweight_dataset/W-1-02_li
```

  - The first column lists the variety for each individual.
  - The second column provides the corresponding absolute path for the input file of Minia.
  - The separator in the table file is a tab character (`\t`).

Since we are using paired-end sequencing data, we need to write a Minia input file for each individual, as shown below:

Plain Text

```
/project1/data/lightweight_dataset/W-2-05_li_R1.fq
/project1/data/lightweight_dataset/W-2-05_li_R2.fq
```

Each row represents a sequencing file corresponding to this individual.

Users may consider using the following command in the paired-end sequencing data directory to auto-generate the Minia input file:

```shell
for file in *_R1.fq; do base_name="${file%_R1.fq}"; realpath "${base_name}_R1.fq" >> "$base_name"; realpath "${base_name}_R2.fq" >> "$base_name"; done
```

The following parameters are optional. Users should adjust them according to their computer configuration:

- `-misap`: Specifies the number of MISA processes to use; the default is 1.
- `-miniap`: Specifies the number of Minia processes to use; the default is 1.
- `-miniac`: Specifies the number of cores for Minia processes; the default is 1.

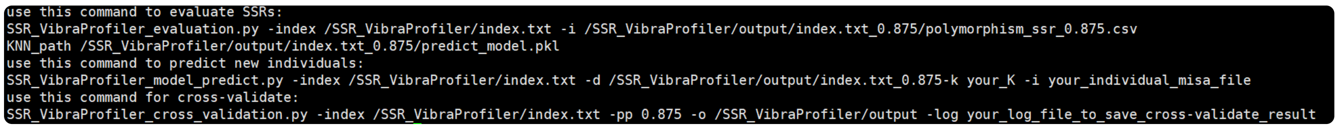
Output information:

If your index file is named `index_file` and your polymorphism indicator is `0.8`, the output files include the following:

- `all_data.csv`: records the numerical features of all SSRs across all individuals.
- `polymorphism_ssr0.8.csv`: records the numerical features of SSRs filtered under a threshold across all individuals.
- `polymorphism_ssr_list.pkl`: a pkl file storing all the filtered SSRs.
- `predict_model.pkl`: a pkl file storing the model for predicting the labels of other individuals.
- `log.txt`: `log.txt` in the current directory records the execution time of each stage.

```shell
output_path/index_file_0.8/
                    └── all_data.csv
                    └── polymorphism_ssr0.8.csv
                    └── polymorphism_ssr_list.pkl
                    └── predict_model.pkl
current directory/
    └── log.txt
```

Additionally, the output information from `SSR_VibraProfiler_model_build.py` includes the usage commands for the other three scripts, with parameters filled in.



```
use this command to evaluate SSRs:
SSR_VibraProfiler_evaluation.py -index /SSR_VibraProfiler/index.txt -i /SSR_VibraProfiler/output/index.txt_0.875/polymorphism_ssr_0.875.csv
KNN_path /SSR_VibraProfiler/output/index.txt_0.875/predict_model.pkl
use this command to predict new individuals:
SSR_VibraProfiler_model_predict.py -index /SSR_VibraProfiler/index.txt -d /SSR_VibraProfiler/output/index.txt_0.875-k your_K -i your_individual_misa_file
use this command for cross-validate:
SSR_VibraProfiler_cross_validation.py -index /SSR_VibraProfiler/index.txt -pp 0.875 -o /SSR_VibraProfiler/output -log your_log_file_to_save_cross-validate_result
```

> **Commands of other 3 scripts with parameters filled in**

Command for testing *Rhododendron* dataset:

```shell
cd /project1/data/
SSR_VibraProfiler_model_build.py -s 1 -e 3 -pp 0.375 -index index.txt -o /project1/output/ -misap 8 -miniap 4 -miniac 2
```

When testing the lightweight dataset, Docker users do not need to prepare the index file. We have embedded a sample index file `index.txt`, located under `/project1/data/` in Docker, which is suitable for our lightweight dataset. For users who do not choose the Docker version, the index files must be prepared manually.

Recognize model evaluation——SSR_VibraProfiler_evaluation.py

This script is used to evaluate the classification performance of the model. It employs three dimensionality reduction methods (PCA, UMAP, and t-SNE) along with K-means clustering, and compares the clustering results with the true labels.

Usage and Options:

```shell
SSR_VibraProfiler_evaluation.py -index -i
```

The following parameters are essential:

- **`-index`** : Specifies the index file, which should be the same as the one used in `SSR_VibraProfiler_model_build.py`.
- **`-i`** : Specifies the input file, which is the `polymorphism_ssr.csv` generated by `SSR_VibraProfiler_model_build.py`.

~~~~~~~~~~~~~~~~~~~~~**Update in 25.2.26**~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The following parameters are optional:

- **`-c`** : Specifies the color, this file contains one RGB color per line. Please ensure that the total number of colors corresponds to the total number of varieties. Here
~~~~~~~~~~~~~~~~~~~~~

is an example:

```Shell
#FF0000
#00FF00
#0000FF
#FFFF00
#800080
#00FFFF
#FFA500
#FFC0CB
```

Output information:

This script outputs clustering result plots (in PDF format) corresponding to the three dimensionality reduction methods (PCA, t-SNE, and UMAP) and K-means cluster.

Command for testing *Rhododendron* dataset:

```Shell
SSR_VibraProfiler_evaluation.py -index /project1/data/index.txt -i /project1/output/index.txt_0.375/polymorphism_ssr_0.375.csv -c color.txt
```

Individual variety identification——SSR_VibraProfiler_model_predict.py

This script is used to predict the label (variety) of an unknown individual.

Usage and Options:

```Shell
SSR_VibraProfiler_model_predict.py -i -d -index -k
```

The following parameters are essential:

- `-i` : Specifies the SSR information file (`.misa` format) for the individual whose variety is to be predicted.
- `-d` : Indicates the folder containing the KNN-based model file and the filtered SSR list, which are the final results from the previous step.

- `-index` : Refers to the same index file used in the first step.
- `-k` : Represents the number of nearest neighbors (*k*) in the KNN prediction.

Output information:

The prediction result is not directly output as a single label. Instead, considering the model may contain varying numbers of individuals per variety, the script outputs the top *k* closest individuals to the one being predicted, based on distance. For example, if `-k 5` is specified, the output will display the five nearest neighbors in order of proximity.

Command for testing *Rhododendron* dataset:

```shell
SSR_VibraProfiler_model_predict.py -index /project1/data/index.txt -d /
project1/output/index.txt_0.375 -k 39 -i W-1-09_li.contigs.fa.misa
```

Cross-validation——SSR_VibraProfiler_cross_validation.py

This script performs leave-one-out cross-validation using an index file as input. During the process, it automatically calls `SSR_VibraProfiler_model_build.py` to construct the identification model and `SSR_VibraProfiler_model_predict.py` to predict the label of the held-out individual. The final output is a file containing the prediction results for all individuals.

Usage and Options:

```shell
SSR_VibraProfiler_cross_validation.py -index -pp -o -log
```

The following parameters are essential:

- **-index**: Refers to the same index file used in the first step.
- **-pp**: Indicates the polymorphism (as described in the main text of the paper).
- **-o**: Output directory path.
- **-log**: File path that saved result data.

Output information:

This script generates a log file in the form of a tab-delimited table, where each row represents the cross-validation result of one individual.

The first column of each row indicates an individual, while the remaining columns list the variety labels of the closest individuals in order of proximity — the second column corresponds to the nearest individual, the third to the next closest, and so on. For example:

Command for testing *Rhododendron* dataset:

Shell

```shell
SSR_VibraProfiler_cross_validation.py -index /project1/data/index.txt -pp 0.375 -o /project1/output -log cross_log.txt
```

About this manual

This version of the manual was finalized on April 18, 2025. If users have any questions, they are welcome to contact us via email at jiangchen1234@163.com. We will continuously update our software and documentation on our GitHub.