SVHIP

May 19, 2021

1 Introduction

Svhip is a software developed in Python for the automatic retraining of the RNAz 2.0 SVM-Classifier, a bioinformatics tool for the de novo discovery of non-coding RNA in whole genome screens. It processes Clustal-Alignments or raw FASTA files and returns a list of feature tuples, that can either be used to recalibrate or test the RNAz software or train a completely independent classifier. Working examples for the latter are included in the repository `https://github.com/chrisBioInf/svhip_Dev`, provided is a Random Forest, a Logistic Regression classifier and others. The RNAz classification software can be downloaded here: `https://www.tbi.univie.ac.at/software/RNAz/#download`.
The basic Svhip-workflow looks something like this:

2 Installation

Will be done when process is clear.

3 Basic Usage

To run Svhip, simply type:

```
$ svhip
```

Since no other arguments are given, this will direct you to the manual page. Take note of the following basic command line arguments:

```
$ svhip -i [INPUT] -o [OUTPUT] [OPTION] [OPTION ARGUMENTS]
```

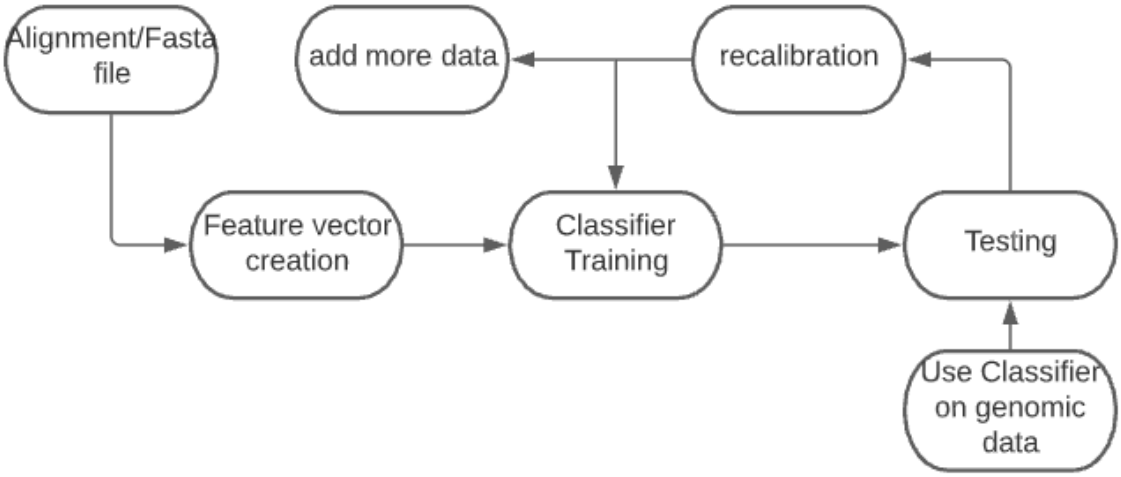

Figure 1: Basic Svhip-workflow

3.1 OPTIONS, core functionality

- -data_gen : Takes the arguments given in [OPTION ARGUMENTS] as well as an [INPUT] parameter and writes a .dat file of training feature vectors based on input files. Please note that datageneration module has its own set of options, for a complete list refer to help function of this module, i.e.:

  ```
  $ -data_gen --help
  ```

  In any case, alignment windows are created and can later be used as testing instance with RNAz.

- -write_m : Takes the arguments given in [OPTION ARGUMENTS] as well as an [INPUT] parameter, optimizes hyper parameters and writes a .model file based on training data gathered using the datageneration module (a .dat file written by the -data_gen or -auto methods). Please note that the write_model module has its own set of options, for a complete list refer to help function of this module, i.e.:

  ```
  $ svhip -write_m --help
  ```

- -auto Combines both the options above. Takes an [INPUT] parameter and creates a .model file. Note that, at the moment, this option will be

executed using the default parameters for both modules, so the input syntax amounts to:

```
$ svhip -i [INPUT] -o [OUTPUT FILE] -auto
```

Alignment windows created are still saved for potential later use.

These three form the core functionality of Svhip. -data_gen will take an alignment file you provide with the -i [alignment name] parameter and calculate a set of corresponding feature vectors for training. These will by default be written to a file calles [alignment name].dat but can be specified with -o [output name], like:

```
$ svhip -i [alignment] -i myoutput.dat -data_gen
```

In the cloned repository, the folder example_data contains a few FASTA-files that can be used for testing. Svhip will automatically determine the input format (clustal or FASTA file).

3.2 File type suffixes

In general, Svhip utilizes four main suffixes to designate different file types used during the process:

- .dat : .dat files contain calculated features of alignments/alignment windows. They are created any time a new input alignment is processed and may contain positive and negative or only positive feature vector instances. These are already scaled and can be directly used for classifier training using the -write_m command line argument and are utilized to hold training data for quick later recalibration or to allow flexible addition of new data. For this reason they are also saved as an intermediate result when following the full pipeline from alignment to trained classifier using the -auto command. Manually editing them is highly deprecated.
- .model : Contain all data necessary for the trained classifier to function and can be simply loaded using the edited RNAz software. Loading a new .model file with RNAz is equivalent to swapping the (dinucleotide, sequence based) classifier. End product of the pipeline.

- .aln_align_n_i : Partial alignments, where n refers to the number of sequences per alignment and i is an internal index counting the number of files created. These contain the alignment windows created from the pre processed input alignment (default columns: 120, step size: 40). From these, feature vectors contained in the .dat files are calculated. They also serve as test set instances, as they are already processed for prediction using RNAz (see RNAz documentation).

- .random : 'Negative' control alignments, created with SISSIz v. 0.11. These serve as dinucleotide and gap-controlled null models for both negative instance creation and structural conservation validation. Remember to also include the randomized alignment windows when constructing a test set with Svhip. For classifier training purposes, they are automatically included upon generation.

Note that these are primarily used to distinguish file types and can be renamed if the need arises. With that in mind, let's return to the usage example. If we want to use the newly generated feature vector file (aln.dat from now on) to train an external classifier, we are already done: We can export the plain text file and use it in any way we want (examples for a python-based parsing is provided in the repository along example alternate classifiers). If we want to use it to train a classifier for RNAz 2.0, we would type:

```
$ svhip -i aln.dat -o aln.model -write_m
```

This will suffice to write all necessary classifier data in the aln.model file once training is finished. This can then immediatly be used with the RNAz version edited for dynamic model loading or by replacing the native dinucleotide_decision.model file before compilation. Please also have a look at the individual –help pages of the data_gen and write_m arguments that further explain individual options, like maximum pairwise sequence identity, depth of crossvalidation during classifier training and others. If you want to create a .model file for RNAz and want to take a shortcut, you can also combine both command explained above, by typing:

3.3 General OPTIONS

- -h,–man : Displays this manual. When combined with the -data_gen, -write _m arguments, it describes specific options for these methods instead. Since -auto combines both methods, [TODO]

- -svcfg : Saves the current parameter string (i.e. arguments except [INPUT] and [OUTPUT] as a <name>.cfg file in /configs. [EXPERIMENTAL] Use at your own discretion.
- -ldcfg : Loads a set of arguments saved as a .cfg file in /configs. Only filename (with or without .cfg) is required. [INPUT] arguments are ignored and will still have to be manually given. [EXPERIMENTAL] Use at your own discretion.

4 Manuals

This section will describe the main methods -data_gen and -write_m in greater detail.

4.1 data_gen

This module accepts one or several (stored in one directory) Rfam Alignments as input and creates raw training data for use in decision model creation. Standard usage:

```
$ svhip -i [INPUT] -o [OUTPUT] -data_gen [OPTIONS]
```

If it is not clear which parameter flags are to be set, a quick example program call would be:

```
$ svhip -i [INPUT] -o meins.dat -data_gen
```

This writes the program output to the new file 'meins.dat'. Here, we simply leave the [OPTIONS] field empty in this case the program will fall back to default parameters. Note that the -readall flag (see below) must be set if you want to read more than one alignment at once (which is usually the case). The -readall option accepts a file listing of Rfam alignments or FASTA files and their respective categories and then writes the extracted data into a single output file. Options that require an input value are used according to schematic:

... -option [value] ...

All others, for example -mute, are simply True/False flags that are set True with:

... -option ...

Also note, that the module can also be used ignoring the main module:

```
$ python3 datageneration.py -i [INPUT] -o [OUTPUT] [OPTIONS]
```

...but this is deprecated. I will not support with resulting issues.

4.1.1 OPTIONS

- -h Displays this manual.

- -mute Default: False
  If flag is set, mutes certain screen outputs, i.e print messages regarding discarded sequences, alignments, program status... Note that some important system messages and the output of the LIBSVM library will still be displayed.

- -minid Default: 50
  min_identity (default: 50):
  Any given sequence is filtered if it has pairwise identity with another sequence < min_identity

- -maxid Default: 98
  Any given sequence is filtered if it has pairwise identity with another sequence > max_identity

- -smpl Default: 100
  How many alignments should be drawn to estimate empirical p-value for structural similarity if simulateAlignment (-sim) is set to False.

- -nproc Default: (Number of CPU cores -1)
  Target number of processes for the multiprocessing pool. Should probably not be manually set, but here it is.

- -o Is used to give chosen name to output file. Name must be next parameter after -outf option,
  i.e ... -outf <name> ...

Important main function calls, also described in the general manual page:

- -readall : Option for reading multiple input alignments at once. Supports and automatically chooses one of two operation modes:

  1) Path to an .ini file with alignment files in same folder is given. Accepts a file with list of .fasta file names and respective categories. The program will read through file until a) end of file or
  b) a '#'
  is encountered. All extracted data points will be written to same output file.

  2) Path to folder with multiple input files meant to be processed. Svhip will ask if negative sets should be auto-generated for all of them.

  ```
  $ svhip -i [INPUT DIRECTORY] -o [OUTPUT] -testset_cr -readall
  ```

- -extract : Accepts an alignment window file [2-12 seqs] (suffix: aln_align_n_i) and returns a tripel of values:

  z-score, SCI, Shannon-Entropy.

  To be used if sorting of input data was done previously. Can quickly scan over values for given files if additional sorting steps are to be implemented... ...but does not contribute much to the actual pipeline right now.

4.2 write_m

This module uses the data prepared by datageneration.py (-data_gen) to train classifiers for use in RNAz prediction. Standard usage:

```
$ svhip -i [INPUT] -o [OUTPUT] -write_m [OPTIONS]
```

If it is not clear which parameter flags are to be set, a minimalist working program call would be:

```
$ svhip -i [.dat INPUT FILE ] -o [.model OUTPUT FILE] -write_m
```

Or, if not using the main module for some reason:

```
$ python3 write_model.py -o [OUTPUT] -i [INPUT]
```

This writes the program output to the new file designated in [OUTPUT], by default a file with the .model suffix (see main man page). Another way would be to simply leave the -o option unflagged - in this case the program will fall back to default parameters (i.e. input filename with a .model added in input directory). All options are used according to schematic:

... -option [value] ...

4.2.1 Options

- -h: Displays this manual.
- -c_low: Default: 0
  Minimum C value for grid search. Has to be ¿= 0
- -c_high:Default: 10
  Maximum C value for grid search.
- -g_low : Default: 0
  Minimum gamma value for grid search.
- -g_high :Default: 10
  Maximum gamma value for grid search.
- -num_c : Default: 10
  Number of C values to generate for grid search. Effectively changes x axis length of the search grid.
- -num_g : Default: 10
  Number of gamma values to generate for grid search. Effectively changes y axis length of the search grid.
- -n_fold :Default: 10
  Number of crossvalidation steps per datapoint in search grid. Influences how trustworthy the calculated mean square error and thereby how accurate the written model is. If runtime is not an issue, higher is probably better.
- -nu : Default: 0.5 The parameter nu is an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. Must be in [0,1]. NOT suggested to be set manually.
- -nproc : Default: Number of CPU cores Sets the target number of parallel processes in grid search to this value.
- -mute : Default: False
  If flag is set, silences certain verbose screen outputs and status reports.

- -o : Destine a file for the output to be written into. If -o is not given ¡inputfilename¿.model will be written in folder 'models'.

5 Svhip workflow

In the most broad sense, Svhip supports three different ways of getting from a set of input alignments to a working classifier to be used with RNAz:

- (1) The most simple case is beginning with a single input fasta file. Obviously, this is rarely sufficient for a functional classifier, but it alloes the flexible later inclusion of one additional alignment file in a pre-existing .dat file, that can then be used to retrain the classifier with the -write_m option without having to reprocess the entire pipeline for all alignments. An example case would be:

  ```
  $ svhip -i [INPUT ALIGNMENT] -o [TARGET .dat FILE] -data_gen
  ```

  Svhip will then ask if a negative instance set should automatically be generated. If "y" is given, the alignment will be processed and negative and positive feature vector instances added to the .dat file (if none is present, it will be generated).
  If "n" is answered, only positive feature vectors will be calculated.
  Should for some reason, a classifier really be trained using only one input alignment, it is sufficient to use:

  ```
  $ svhip -i [INPUT ALIGNMENT] -o [OUTPUT .model FILENAME] -auto
  ```

- (2) The typical use case is assumed to be as follows:
  A collection of input .FASTA files is to be processed, filtered and then a classifier treained based on their features, with automatic negative set creation. For this case, it is sufficient to include all input files in a single directory and tell Svhip this directory using the -auto and -readall command line argument:

  ```
  $ svhip -i [INPUT DIRECTORY] -o [OUTPUT .model FILENAME] -auto -readall
  ```

  Then 'y' should be answered when Svhip asks if negative sets should automatically be generated for all input files. A .model file will be written using both positive and generated negative instances.

- (3) If for any reason the automatic generation of negative instances is not desired, Svhip can be given directions to a list of files to include in

either the negative or the positive training set. This .ini file has to be configured as follows:

+1:[FILEPATH]
-1:[FILEPATH]
...

where +1 denotes a path to a file to include in the positive training set and -1 the path to a negative set entry. The function call has then to refer to the .ini file:

```
$ svhip -i [FILE.ini] -o [OUTPUT .model FILE] -auto -readall
```

Or, if a testset is to be created this way:

```
$ svhip -i [FILE.ini] -o [OUTPUT .dat FILENAME] -data_gen -readall
```

- ((4)) Not starting from an input FASTA file or alignment, to process a pre created .dat file (for example using the (1) option described here) to train a classifier is done by simply using the following command:

  ```
  $ svhip -i [.dat INPUT FILE] -o [.model OUTPUT FILE] -write_m
  ```

  NOTE: For any of these use cases, the explicit naming of the output file can be omitted. In this case, a name will be generated based on the input file name, and the output file will be generated in the current working directory!

  Now, the above cases all assume that default parameters are to be used. For explicit descriptions of all arguments to fine-tune parameters like number of cross-validation steps, sample size for secondary structure validation and C and gamma parameter range refer to the -data_gen and -write_m man pages (see above). Parameters from both pages can be used with the -auto command as it combines both methods.

6 Processing multiple files at once

Obviously, only appending data to a .dat file alignment per alignment is tedious and we usually want to process many alignments, that we previously selected, at once. The solution for this is the aln.model readall flag for the aln.model data_gen program. It can be used like this:

```
$ svhip -i [inputfile] -o [outputdirectory] -data_gen -readall
```

Svhip will expect a folder with either FASTA files or clustal alignments as input and then process all found files one by one, writing the output to the given .dat file.

7 Contact

Christopher Klapproth
christopher@bioinf.uni-leipzig.de
University Leipzig
Interdisciplinary Centre for Bioinformatics

8 License