

Svhip software for retrainable identification of conserved genes in multiple genome alignments

Christopher Klapproth

01.08.2022

Contents

1	Introduction	3
2	Installation	3
3	Running tests	4
4	Basic usage	4
4.1	Training Data generation	6
4.1.1	Example	7
4.1.2	Hexamer models	10
4.1.3	Arguments	11
4.2	Combination of generated feature files	13
4.2.1	Arguments	13
4.3	Model training	15
4.3.1	Example	16
4.3.2	Arguments	16
4.4	Model evaluation	19
4.4.1	Example	19
4.4.2	Arguments	19
4.5	Feature calculation of genome alignments	20
4.5.1	Example	20
4.5.2	Arguments	21
4.6	Prediction	22

4.6.1	Example	22
4.6.2	Arguments	22
4.7	Concluding remarks	24

1 Introduction

Svhip is a software developed in Python 3.8 for analysis of multiple genome alignments in MAF format for the identification of conserved functional gene sites. It provides options for the search for both protein coding sequences (CDS) as well as the identification of evolutionary conserved secondary structures, hinting at functional non-coding sequences. A core feature of Svhip is the possibility to freely retrain the classifier to account for different genomic contexts, usually done by providing preselected training examples in the form of ClustalW-alignments. Some of its features directly build on the RNAz framework (<https://www.tbi.univie.ac.at/software/RNAz/#download>) for the identification of secondary structure sites of high conservation, with the core difference being the unchangeability of the underlying RNAz model and its lack of support for the identification of coding sequences.

2 Installation

In terms of external requirements, Svhip will require a working `perl` installation and the installation of the software ClustalW2. All needed python libraries are contained in the included conda environment and we suggest using it for the installation of these dependencies. We suggest installation using conda and a new environment:

```
$ conda create --name svhip_env python=3.9
```

which will generate a new conda environment using python version 3.9. Switch to the new environment:

```
$ conda activate svhip_env
```

Then we install Svhip from the bioconda channel using:

```
$ conda install -c bioconda svhip
```

This should download and install all required files. We will verify the installation in the next step.

-

From inside this environment, programs associated with the Svhip framework can be safely executed without interfering with other user-specified libraries installed on the machine. To test functionality of the framework itself, proceed to the following section.

3 Running tests

To test the installation and the conda environment, you can call the internal test by typing

```
$ svhip check
```

In case of successful installation, this should produce the following output to the screen:

```
TEST 1 / 5: SUCCESS
TEST 2 / 5: SUCCESS
TEST 3 / 5: SUCCESS
TEST 4 / 5: SUCCESS
TEST 5 / 5: SUCCESS
Program ran for 17.77 seconds.
```

This means that all individual subroutines work as intended in a simple test scenario. In the following section we will now take a look at usage of individual program parts.

4 Basic usage

To run Svhip, simply type:

```
$ svhip
```

which will bring up the help menu. Svhip supports different modes of operation corresponding to it's different uses from initial training data generation, to calculation of features in alignment windows to final prediction. These are generally called from the command line as follows

```
$ svhip [PROGRAM] [OPTIONS]
```

where PROGRAM refers to the name of the subprogram to be called. `--help` statements are available for each of these. All these subprograms are listed as follows and will be explained in more detail below:

- data
- combine
- training
- evaluate
- features
- predict

4.1 Training Data generation

The data generation program, called in the simplest case using

```
$ svhip data -i [INPUTFILE] -o [OUTPUT FOLDER]
```

serves to preprocess either ClustalW alignments or collections of sequences in Fasta files and calculates vectors of features used in further classifier training or evaluation. Abstracted, the sequence of taken steps is as follows: Sequences are realigned (if not already aligned) using the ClustalW2 installation located on the users machine. Then, using the (integrated) `rnazSelectSeqs.pl` script of the RNAz framework, subset of sequences optimized for average pairwise identity is selected (Default: up to 100). If less sequences than the maximum specified with the `--num-sequences` parameter are in the alignment and none of them surpass the identity threshold specified with the `--max-id` parameter, all are retained.

This alignment is then sliced in overlapping windows of sub-alignments with between 2 and 12 sequences in each per default. Usually multiple of these alignment windows are generated per number of sequences. Once these are generated, a feature vector consisting of Structural conservation index, z-score of Minimum free energy (MFE), Shannon-entropy, alignment-wide Hexamer score and Codon conservation score is calculated and written to output as a tab-delimited table `.tsv` file.

A special feature of Svhipe is the possibility to automatically generate a fitting negative training set based on the input data. This is achieved using either the `rnazRandomizeAln.pl` tool or the `SISSIZ` software for dinucleotide-controlled null models. For the latter, an installation of `SISSIZ 0.1.1` must be present on the users machine. The negative set generation is initiated using

```
$ svhip data -i [INPUTFILE] -o [OUTPUT FOLDER] --generate-control True .
```

Should the more sophisticated `SISSIZ` based simulation of control data be used, deactivate the default shuffling with

```
$ svhip data -i [INPUTFILE] -o [OUTPUT FOLDER] --generate-control True --shuffle-control False .
```

Otherwise an own negative training set can also be supplied with the `-N`, `--negative` parameter.

In principal the process is equal for the generation of training data based on alignments of coding sequences. However, note that in this case the flag (`-p`) for identification as protein coding has to be set with

```
$ svhip data -i [INPUTFILE] -o [OUTPUT FOLDER] -p CDS .
```

Equivalent to this, training data can also be specifically designated as non-coding with

```
$ svhip data -i [INPUTFILE] -o [OUTPUT FOLDER] -p ncRNA .
```

This is however not necessary for most cases, as a structurally conserved ncRNA type set is assumed by default. Note further that automatic generation of negative training sets in a coding context is supported in principal, but might lead to unexpected behavior in certain cases, as native signals might not be disrupted evenly.

4.1.1 Example

In the installation included is the `/Example` folder, containing an alignment of 45 highly conserved tRNA sequences as derived from the Rfam data base. To test the basic functionality of Svhip, simply navigate to the installation folder and type

```
$ svhip data -i Example/tRNA test.fa -o tRNA test -
```

which will generate an output folder containing several graphical evaluations, a subfolder `Example/tRNA_test_windows` containing sliced alignment windows and the `tRNA_test_trainingdata.tsv` which contains calculated feature vectors. Opening and studying it will reveal that it only contains examples directly generated from the supplied input file, thus, lacking a control set, not being useful for a real classification problem. The generated file should look something like the following Figure 1.

We can let Svhip generate a negative training set for us by typing instead

```
$ svhip data -i Example/tRNA test.fa -o tRNA test --generate-control True
```

File	Edit	Search	View	Document	Help						
	SCI	z-score	of MFE	Shannon-entropy	Hexamer	Score	Codon conservation	Class			
0	1.0085	-3.2421	0.3219	0.35483690460389766	0.6976	RNA					
1	1.092	-3.784	0.4527	0.09685745445270723	0.2382	RNA					
2	1.092	-3.784	0.4527	0.09685745445270723	0.2382	RNA					
3	1.0085	-3.2421	0.3219	0.35483690460389766	0.6976	RNA					
4	1.092	-3.784	0.4527	0.09685745445270723	0.2382	RNA					
5	1.0217	-2.2574	0.2703	0.9657658245527562	0.4482	RNA					
6	1.0085	-3.2421	0.3219	0.35483690460389766	0.6976	RNA					
7	1.0217	-2.2574	0.2703	0.9657658245527562	0.4482	RNA					
8	1.0997	-3.0004	0.5402	0.21396822096691526	0.3372	RNA					
9	1.1083	-2.1978	0.5079	0.9171772123475569	0.4004	RNA					
10	1.1083	-2.1978	0.5079	0.9171772123475569	0.4004	RNA					
11	1.1083	-2.1978	0.5079	0.9171772123475569	0.4004	RNA					
12	1.1083	-2.1978	0.5079	0.9171772123475569	0.4004	RNA					
13	1.1483	-3.2156	0.606	0.14373802308995912	0.2407	RNA					
14	1.0997	-3.0004	0.5402	0.21396822096691526	0.3372	RNA					
15	1.1494	-3.2988	0.6721	0.11944454040812273	0.2382	RNA					
16	1.1332	-2.7497	0.6223	0.8146307436334396	0.3372	RNA					
17	1.1533	-3.0097	0.6736	0.025348378510908574	0.2407	RNA					
18	1.1332	-2.7497	0.6223	0.8146307436334396	0.3372	RNA					

Figure 1: Output file after running the data generation command on an example alignment of highly conserved tRNA sequences.

This process will take a little longer. Looking at the output file again we can now see that there are feature vectors labeled as either `ncRNA` or `OTHER`, the latter here referring to training instances simulating an unspecific (i.e. for example intergenic) genomic background. Figure 2 shows how the generated vectors for the generated control could look like.

In this context, the automatically generated graphical evaluations which serve to illustrate distribution of features between classes become more interesting (see Figure 3).

File	Edit	Search	View	Document	Help		
153	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
154	-0.0	-0.096	0.9949	0.467557925459762	0.401	OTHER	
155	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
156	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
157	-0.0	-0.096	0.9949	0.467557925459762	0.401	OTHER	
158	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
159	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
160	-0.0	-0.0996	0.88	0.3708121887134999	0.2461	OTHER	
161	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
162	-0.0	0.018	1.0045	0.4655722521162211	0.2075	OTHER	
163	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
164	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
165	-0.0	0.018	1.0045	0.4655722521162211	0.2075	OTHER	
166	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
167	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
168	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
169	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
170	-0.0	-0.0702	0.947	0.3797018381888995	0.2785	OTHER	
171	-0.0	-0.1269	0.9844	0.39761217433083984	0.3695	OTHER	
172	-0.0	-0.2555	0.9744	0.3071854170270574	0.2784	OTHER	
173	-0.0	-0.1269	0.9844	0.39761217433083984	0.3695	OTHER	
174	-0.0	-0.1269	0.9844	0.39761217433083984	0.3695	OTHER	

Figure 2: Output file containing automatically generated negative training instances created with the data generation subprogram on an example alignment of highly conserved tRNA sequences.

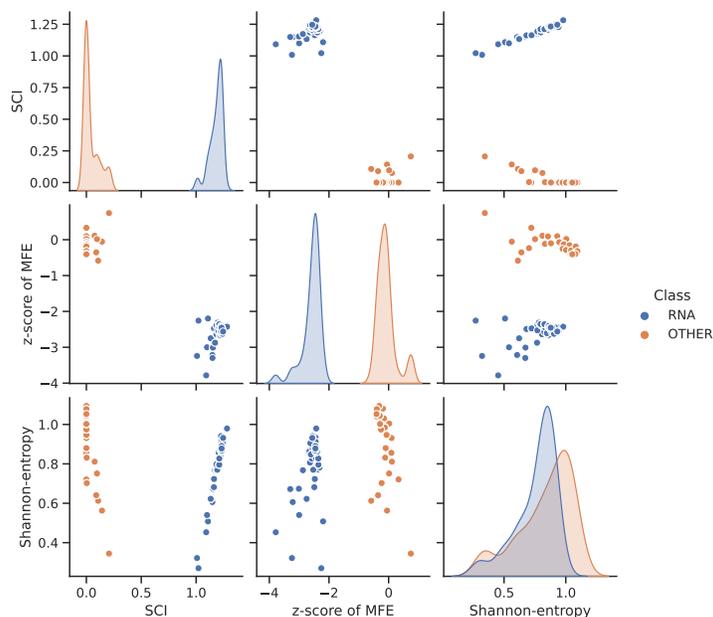


Figure 3: Features as calculated from generated alignment windows of a sample of aligned tRNA sequences. As can be seen there is strong differentiation between the native alignments and the generated control set.

Another point of note here is the integrated filtering mechanic for selection of only statistically significant alignment windows. Based on native alignment windows, randomized alignments are generated and their pair-wise tree edit distances used as an approximation of secondary structure difference. From the average tree edit distance of these alignments is a distribution approximated, which is used as a filter to only select statistically significant alignment windows for feature calculation. An example of these distributions and their overlap can be reviewed in Figure 4. Should this property not be desired for some reason, it can be turned off with

```
$ svhip data -i Example/tRNA test.fa -o tRNA test --no-structural-filter True
```

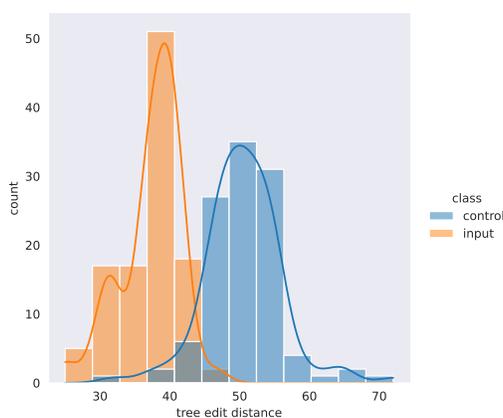


Figure 4: Distributions of average pair-wise tree edit distances of secondary structure representations in native alignment windows sliced from an alignment of tRNA sequences and the corresponding control set.

4.1.2 Hexamer models

For the calculation of the alignment-wide hexamer score a heuristic is employed that builds on a preexisting model assigning each possible 6-mer of nucleotides a corresponding frequency in a coding and non-coding context. If not otherwise specified, a general model based on Human training data will be employed. As genomic contexts differ in their distribution of hexamers, a customized Hexamer model should be provided in other cases:

```
$ svhip data -i [INPUT] -o [OUTPUT] -H [HEXAMER MODEL FILE]
```

In principal these models are tab-delimited files of 4096 lines that contain one possible hexamer per line followed by an empirical probability to find this constellation in a coding (first) or non-coding environment. Given a genome and a known .gtf annotation file with coding regions clearly marked as CDS, this model can be recalibrated using the create_hexamer_model.py script. For further reference see also the CPAT software, employing this property in a single-sequence context [CITE].

4.1.3 Arguments

Usage:

```
svhip data [options]
```

Options:

```
--version          show program's version number and exit
```

```
-h, --help        show this help message and exit
```

```
-i IN_FILE, --input=IN_FILE
```

The input directory or file (Required).

```
-o OUT_FILE, --outfile=OUT_FILE
```

Name for the output directory (Required).

```
-N NEGATIVE, --negative=NEGATIVE
```

Should a specific negative data set be supplied for data generation? If this field is EMPTY it will be auto-generated based on the data at hand (This will be the desired option for most uses).

```
-d MAX_ID, --max-id=MAX_ID
```

During data preprocessing, sequences above identity threshold (in percent) will be removed. Default: 95.

```
-n N_SEQS, --num-sequences=N_SEQS
```

Number of sequences input alignments will be optimized

towards. Default: 100.

-l WINDOW_LENGTH, --window-length=WINDOW_LENGTH
Length of overlapping windows that alignments will be sliced into. Default: 120.

-s SLIDE, --slide=SLIDE
Controls the step size during alignment slicing and thereby the overlap of each window.

-w N_WINDOWS, --windows=N_WINDOWS
The number of times the alignment should be fully sliced in windows - for variation.

-g GENERATE_CONTROL, --generate-control=GENERATE_CONTROL
Flag to determine if a negative set should be auto-generated (Default: False).

-c SHUFFLE_CONTROL, --shuffle-control=SHUFFLE_CONTROL
Use the column-based shuffling approach provided by the RNAz framework instead of SISSIz (Default: False).

-p POS_LABEL, --positive-label=POS_LABEL
The label that should be assigned to the feature vectors generated from the (non-control) input data. Can be CDS (for protein coding sequences) or ncRNA. (Default: ncRNA).

-H HEXAMER_MODEL, --hexamer-model=HEXAMER_MODEL
The Location of the statistical Hexamer model to use. An example file is included with the download as Human_hexamer.tsv, which will be used as a fallback.

-S STRUCTURE_FILTER, --no-structural-filter=STRUCTURE_FILTER
Set this flag to True if no filtering of alignment windows for statistical significance of structure

4.2 Combination of generated feature files

The `Svhip combine` command may be used to unite several independently generated files containing feature vectors. This small subprogram mostly serves for quick testing of different data selection approaches. Usage is simply typing

```
$ svhip combine -i [DIRECTORY] -o [OUTPUT] --prefix [PREFIX]
```

where the `-o` argument denotes simply the name or path to the combined output `.tsv` file. Note that for this subprogram the otherwise mandatory `-i` input argument is not needed and if none is provided, the current working directory will be scanned. Otherwise it should point to a directory containing all the previously generated files one wishes to include in the combination. The `-p`, `--prefix` argument serves to indicate a mandatory prefix that all files have to share before they are included, this can be used to sort out generated feature files from different origins, for example. As an example, if `Test` is passed to the `--prefix` argument, only files starting their name with `Test` will be included.

4.2.1 Arguments

Usage:

```
svhip combine [options]
```

Options:

```
--version          show program's version number and exit  
-h, --help         show this help message and exit
```

```
-i IN_FILE, --input=IN_FILE
```

The input directory or file (Required).

```
-o OUT_FILE, --outfile=OUT_FILE
```

Name for the output directory (Required).

```
-p PREFIX, --prefix=PREFIX
```

Prefix for selection of files to combine. For example, if set to `TEST`, only valid feature vector containing

files with the prefix TEST will be added.

4.3 Model training

Having a file of training data in `.tsv` format as generated in the previous step, training a new model is in principal as simple as typing out

```
$ svhip training -i [INPUT] -o [OUTPUT] .
```

There are however here a few things to consider. The first aspect is the ability of Svhip to harness the `sklearn` library to generate different kinds of models, namely Random Forest (RF), Logistic Regression (LR) and Support Vector Machine Models (SVM), with the latter being the default case. In our tests there seems to be no model type that excels on all different kinds of input data, which is why we leave the decision up to the expertise of the end user. It should be noted that in terms of evaluation and prediction purposes all of them are treated equally. Selecting a different type of model can be done with

```
$ svhip training -i [INPUT] -o [OUTPUT] -M RF
```

to for example select the Random Forest classifier using the `-M` parameter.

The second core aspect is the integrated option for optimization of hyperparameters either by grid search or by a random walk approach. By default, optimization will be turned on within a reasonable range of base parameters, that are fully customizable. However, which parameters can be selected per optimizer is dependent on the model in use. The SVM classifier type, employing the Cost and gamma parameters, comes with the option to set the following parameters

```
$ svhip training -i [INPUT] -o [OUTPUT] --low-c 1 --high-c  
1000 --low-gamma 1 --high-gamma 1000 --hyperparameter-steps 10
```

which indicates the usage of min values 1 and max values 1000 (indicated by the `--min` parameters) for both hyperparameters. Furthermore, we decide on trying up to 10 values for each (indicated by the `--hyperparameter-steps` argument) which will be spread out linearly, thus creating a search grid of 100 value pairs in case a grid search is used. Instead of a linear succession of values, a log scale can also be defined by setting the `--logscale` flag to `True`.

4.3.1 Example

Using the training data file based on tRNA as generated in the previous section to train an SVM classifier with hyperparameter optimization can be achieved with:

```
$ svhip training -i tRNA test trainingdata.tsv -o tRNA model -  
  --model SVM --optimize-hyperparameters True .
```

This will write both a tRNA_model.model file as well as corresponding parameters file containing the values used in normalization of parameters. The model file can then be used in further steps for prediction purposes, or can be first evaluated against a known data set as described in the next section.

4.3.2 Arguments

Usage:

```
svhip training [options]
```

Options:

```
--version          show program's version number and exit  
-h, --help        show this help message and exit
```

```
-i IN_FILE, --input=IN_FILE
```

The input directory or file (Required).

```
-o OUT_FILE, --outfile=OUT_FILE
```

Name for the output directory (Required).

```
-S STRUCTURE, --structure=STRUCTURE
```

Flag determining if only secondary structure conservation features should be considered. If True, protein coding features will be included (Default: False).

```
-M ML, --model=ML      The model type to be trained. You can choose LR  
(Logistic regression), SVM (Support vector machine) or  
RF (Random Forest). (Default: SVM)
```

`--optimize-hyperparameters=OPTIMIZE`
Select if a parameter optimization should be performed for the ML model. Default is on.

`--optimizer=OPTIMIZER`
Select the optimizer for hyperparameter search. Search will be conducted with 5-fold crossvalidation and either of 'gridsearch' (default, more precise) or 'randomwalk' (faster).

`--low-c=LOW_C` SVM hyperparameter search: Lowest value of the cost (C) parameter to optimize. Does nothing if no SVM classifier is used.

`--high-c=HIGH_C` SVM hyperparameter search: Highest value of the cost (C) parameter to optimize. Does nothing if no SVM classifier is used.

`--low-gamma=LOW_G` SVM hyperparameter search: Lowest value of the gamma parameter to optimize. Does nothing if no SVM classifier is used.

`--high-gamma=HIGH_G` SVM hyperparameter search: Highest value of the gamma parameter to optimize. Does nothing if no SVM classifier is used.

`--hyperparameter-steps=GRID_STEPS`
Number of values to try out for EACH hyperparameter. Values will be evenly spaced. Default: 10

`--logscale=LOGSCALE` Flag that decides if a logarithmic scale should be used for the hyperparameter grid. If set, a log base can be set with `--logbase`.

`--logbase=LOGBASE` The logarithmic base if a log scale is used in hyperparameter search. Default: 10.

`--min-trees=LOW_ESTIMATORS`

Random Forest hyperparameter search: Minimum number of trees before optimization. Does nothing if no RF classifier is used.

`--max-trees=HIGH_ESTIMATORS`

Random hyperparameter search: Maximum number of trees before optimization. Does nothing if no RF classifier is used.

`--min-samples-split=LOW_SPLIT`

Random Forest hyperparameter search: Minimum number of samples for splitting an internal node in the forest. Does nothing if no RF classifier is used.

`--max-samples-split=HIGH_SPLIT`

Random hyperparameter search: Maximum number of samples for splitting an internal node in the forest. Does nothing if no RF classifier is used.

`--min-samples-leaf=LOW_LEAF`

Random Forest hyperparameter search: Minimum number of samples for splitting a leaf node in the forest. Does nothing if no RF classifier is used.

`--max-samples-leaf=HIGH_LEAF`

Random hyperparameter search: Maximum number of samples for splitting a leaf node in the forest. Does nothing if no RF classifier is used.

4.4 Model evaluation

This subprogram allows for the evaluation of a generated model file by analyzing accuracy, recall rates and the tradeoff between False positive rate and True positive rate as visualized by a ROC curve.

4.4.1 Example

Usage with the above generated model on the already generated training set would be initiated as follows:

```
$ svhip evaluate -M tRNA model.model -o evaluation test -i -  
tRNA test trainingdata.tsv .
```

The `-M` parameter here serves to indicate the path to the model to be evaluated. Notice that in most cases evaluating the trained model with the training data used to generate this model is not the best option and is only done here for illustrative purposes.

4.4.2 Arguments

Usage:

```
svhip evaluate [options]
```

Options:

```
--version          show program's version number and exit  
-h, --help        show this help message and exit
```

```
-i IN_FILE, --input=IN_FILE
```

The input directory or file (Required).

```
-o OUT_FILE, --outfile=OUT_FILE
```

Name for the output directory (Required).

```
--model-path=MODEL_PATH
```

If running a model test, this is the path of the model to evaluate. The data set to use should be handed over with `-i, --input`.

4.5 Feature calculation of genome alignments

This program calculates the sets of features for MAF genomic alignments cut in overlapping windows using `rnazWindows.pl` or a similar tool. Its basic usage is called as follows:

```
$ svhip.py features -i [ALIGNMENTS] -o [OUTPUT FILE] .
```

Two things are worth considering here. First, the Hexamer score feature calculated from the alignments here is obviously just as dependent on the provided background distribution of 6-mers as for the training data construction. Thus, also here it is advisable to provide a Hexamer model file, using the `-H` parameter:

```
$ svhip.py features -i [ALIGNMENTS] -o [OUTPUT FILE] -H [HEXAMER MODEL]
```

The second aspect is the reading direction in which the genome is supposed to be analyzed. By default, features will be calculated for exactly the aligned sequences present in the supplied input file. However, in many cases functional genes may as well be encoded by the reverse complement strand. If the reverse direction should be calculated as well, we will also have to set the corresponding flag `-R`, `--reverse` to `True`:

```
$ svhip.py features -i [ALIGNMENTS] -o [OUTPUT FILE] -R True .
```

Another property of note for this subprogram is that it automatically attempts to read out genome information from the provided file if possible. If the MAF file contains information regarding genomic coordinates, i.e. start, end and length of the sequence, these will be reflected in the output file as well, along with reading directions.

4.5.1 Example

We can employ the following command to test this subprogram on the MAF file provided in the `/Example` folder. It contains alignment windows from chromosome 1 of an alignment of plant genomes with *Arabidopsis thaliana*.

```
$ svhip.py features -i Example/Arabidopsis_1.maf -o Arabidopsis_1.tsv -R True .
```

This will calculate features for prediction for each of these alignment windows in both forward and reverse direction. The output file should look something like in Figure 5.

	SCI	z-score	of	MFE	Shannon-entropy	Hexamer	Score	Codon conservation	start	end	direction	chromosome
0	0.5641	1.0399	0.16	-0.012215530258095408	0.9478	6755	6830	forward	1			
1	0.1589	0.5788	0.16	0.246138399104788	0.828	6755	6830	reverse	1			
2	0.3613	1.1157	0.3947	1.0066328469758297	0.0184	7154	7226	forward	1			
3	0.1836	1.4161	0.3947	0.5366363765763964	1.145	7154	7226	reverse	1			
4	0.7117	-0.6553	0.1416	0.13711275633546863	1.0402	8058	8125	forward	1			
5	0.2607	0.6407	0.1416	-0.07809306872660932	0.4904	8058	8125	reverse	1			
6	0.9392	-3.3104	0.1949	0.93091980674317	0.7341	309270	309361	forward	1			
7	0.9618	-1.8294	0.1949	1.0097093476590269	1.0288	309270	309361	reverse	1			

Figure 5: Output file after running the `features` command on a set of MAF genomic alignments, prepared for prediction with a trained classifier.

4.5.2 Arguments

Usage:

`svhip features [options]`

Options:

`--version` show program's version number and exit

`-h, --help` show this help message and exit

`-i IN_FILE, --input=IN_FILE`

The input directory or file (Required).

`-o OUT_FILE, --outfile=OUT_FILE`

Name for the output directory (Required).

`-R REVERSE, --reverse=REVERSE`

Also scan the reverse complement when calculating features.

`-H HEXAMER_MODEL, --hexamer-model=HEXAMER_MODEL`

The Location of the statistical Hexamer model to use.

An example file is included with the download as

`Human_hexamer.tsv`, which will be used as a fallback.

4.6 Prediction

For the prediction process, two components are needed. First, a trained model is required to classify previously calculated feature vectors into categories (ncRNA, CDS or OTHER). Secondly, a .tsv file with the feature vectors from input alignments for classification as generated in the previous section is needed. Prediction is initiated using the following command:

```
$ svhip predict -i [FILE WITH FEATURE VECTORS] -o [OUTPUT FILE]
  --model-path [MODEL FILE] --column-label [NAME] .
```

A few notes on the structure of this command: `--model-path` should point to the exact path to a previously trained model. `--column-label` refers to the name of the column in the output file in which the classification results will be stored. So, if we enter "Predictions", the output .tsv will contain a column named "Predictions" with all the assigned labels.

4.6.1 Example

Putting it all together, we can predict the calculated feature vectors from the previous section with the simple model we trained on the tRNA data. For this, we type out:

```
$ svhip predict -i Arabidopsis 1.tsv -o Arabidopsis 1.svhip -
  --model-path tRNA model.model --column-label Prediction .
```

Take a look at the resulting file. It should contain all the information previously contained in the Arabidopsis_1.tsv plus the "Prediction" column. This column should contain many OTHER predictions and one particular window (in both directions) being classified as ncRNA, as seen in Figure 6.

Unnamed: 0	SCI	z-score of MFE	Shannon-entropy	Hexamer Score	Codon conservation	start	end	direction	
chromosome	Prediction								
0	0.5641	1.0399	0.16	-0.0122155302580954	0.9478	6755	6830	forward 1	OTHER
1	0.1589	0.5788	0.16	0.246138399104788	0.828	6755	6830	reverse 1	OTHER
2	0.3613	1.1157	0.3947	1.0066328469758297	0.9184	7154	7226	forward 1	OTHER
3	0.1836	1.4161	0.3947	0.5366363765763964	1.145	7154	7226	reverse 1	OTHER
4	0.7117	-0.6553	0.1416	0.1371127563354686	1.0402	8058	8125	forward 1	RNA
5	0.2607	0.6407	0.1416	-0.0780930687266093	0.4904	8058	8125	reverse 1	OTHER
6	0.9392	-3.3104	0.1949	0.93091980674317	0.7341	309270	309361	forward 1	RNA
7	0.9618	-1.8294	0.1949	1.0097093476590269	1.0288	309270	309361	reverse 1	RNA

Figure 6: Output of the `prediction` subroutine, assigning classification labels to previously calculated feature vectors. The last window from position 309.270 to 309.361 is indeed overlapping with the annotation of a tRNA gene in the AraPort11 annotation.

Indeed, this particular window was added containing an alignment of plant tRNA genes. Although these were not represented in the training set in particular, the high level of secondary structure conservation indicates the presence of some biological function to be annotated here.

4.6.2 Arguments

Usage:

`svhip predict [options]`

Options:

`--version` show program's version number and exit

`-h, --help` show this help message and exit

`-i IN_FILE, --input=IN_FILE`

The input directory or file (Required).

`-o OUT_FILE, --outfile=OUT_FILE`

Name for the output directory (Required).

`-M MODEL_PATH, --model-path=MODEL_PATH`

If running a model prediction

(`predict`), this is the path of the model to

evaluate. The data set to use should be handed over

with `-i, --input`.

`--column-label=PREDICTION_LABEL`

Column name for the prediction in the output.

`--structure=NCRNA` Set to True if only features for conservation of secondary structure should be used. Depends on type of model.

4.7 Concluding remarks

The Svhip software provides a simple and reproducible way to train and utilize classifiers for the identification of evolutionarily conserved protein coding and non-coding genes in screens of multiple genome alignments. We illustrate here only the basic functionality of the tool, demonstrating a simple use case with merely one training example. In practice, it is advisable to construct and evaluate diverse training sets incorporating many different examples of well-conserved and established RNAs before attempting a genome-wide screen for the purpose of *de novo* discovery of functional genes.

We further remark that in any case it is advisable to cross-compare results provided by the Svhip framework with predictions and evaluations provided by other methods, such as mapping predicted genetic loci to transcriptome libraries with the goal of identifying actual expression of predicted genes. Furthermore, we acknowledge that while conservation on evolutionary time scales is undeniably an important aspect in mapping out potential biological function, there are just as undeniably biologically active nucleotide chains that show no such conservation at all. Thus, the approach outlined here will remain blind to these.

List of Figures

1	Output file after running the data generation command on an example alignment of highly conserved tRNA sequences. . . .	8
2	Output file containing automatically generated negative training instances created with the data generation subprogram on an example alignment of highly conserved tRNA sequences. . .	9
3	Features as calculated from generated alignment windows of a sample of aligned tRNA sequences. As can be seen there is strong differentiation between the native alignments and the generated control set.	9
4	Distributions of average pair-wise tree edit distances of secondary structure representations in native alignment windows sliced from an alignment of tRNA sequences and the corresponding control set.	10
5	Output file after running the features command on a set of MAF genomic alignments, prepared for prediction with a trained classifier.	21
6	Output of the prediction subroutine, assigning classification labels to previously calculated feature vectors. The last window from position 309.270 to 309.361 is indeed overlapping with the annotation of a tRNA gene in the AraPort11 annotation.	23