ssHMM Documentation

Release 1.0.0.b3

David Heller

Apr 09, 2020

Contents:

ssHMM stands for “Sequence-Structure Hidden Markov Model”. It is an RNA motif finder and recovers sequence-structure motifs from RNA-binding protein data.

CHAPTER 1

Background

RNA-binding proteins (RBPs) play a vital role in the post-transcriptional control of RNAs. They are known to recognize RNA molecules by their nucleotide sequence as well as their three-dimensional structure. ssHMM combines a hidden Markov model (HMM) with Gibbs sampling to learn the joint sequence and structure binding preferences of RBPs from high-throughput RNA-binding experiments, such as CLIP-Seq. The model can be visualized as an intuitive graph illustrating the interplay between RNA sequence and structure.

CHAPTER 2

Scope

ssHMM was developed for the analysis of data from RNA-binding assays. Its aim is to help biologists to derive a binding motif for one or a number of RNA-binding proteins. ssHMM was written in Python and is a pure command-line tool.

CHAPTER 3

License

The project is licensed under the GNU General Public License.

3.1 Installation

3.1.1 Installation with Conda

Prerequisites

- Conda (https://conda.io)

Preparation of conda evironment

Conda provides isolated environments in which we can install software packages to run separately from other software installed on the system. ssHMM depends on several other software packages which we will install into a brand new conda environment in the next steps.

1. Prepare the *.condarc* configuration file in your home directory (create if it does not exist)

```
  channels:
- bioconda
- defaults
- conda-forge
```

This step makes sure that the required dependencies are downloaded and installed from the correct source repositories (i.e. conda channels). Conflicts have been reported between software packages from different conda channels.

2. Create a new conda environment *sshmm_env* with all required software packages

```
conda create --name sshmm_env python=2.7 pip numpy forgi=1.1 graphviz pygraphviz␣
↪rnashapes=2.1.6 rnastructure ghmm
```

3. Find out in which directory the new environment was created in

```
conda env list | grep sshmm_env
```

4. Make sure that an important environment variable is set for the RNAstructure prediction tool every time the environment is activated. Replace <ENVDIR> with the directory you found out in the previous step.

```
cd <ENVDIR>
mkdir -p ./etc/conda/activate.d
mkdir -p ./etc/conda/deactivate.d
echo -e '#!/bin/sh\n\nexport DATAPATH=<ENVDIR>/share/rnastructure/data_tables/' > ./
↪etc/conda/activate.d/env_vars.sh
echo -e '#!/bin/sh\n\nexport DATAPATH=' > ./etc/conda/deactivate.d/env_vars.sh
```

Installation of ssHMM

5. Activate the new conda environment

```
source activate sshmm_env
```

6. Install ssHMM

```
pip install sshmm
```

3.1.2 Installation via Docker

Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

To obtain the ssHMM Docker image and run it, follow these steps:

1. Install Docker on your platform as described at `https://docs.docker.com/engine/getstarted/`
2. Run Docker and check whether everything is working:

```
docker version
```

3. Run the Docker image containing ssHMM. The following command will also download the image if it cannot be found locally:

```
docker run -it hellerd/sshmm
```

A running image is called a container. Once the container has been started, you can access it via a command line interface. You can now run ssHMM, e.g. by typing `train_seqstructhmm --help`. You can exit the container with `exit`.

Hint: By default a container's file system persists even after the container exits. This makes debugging a lot easier and you retain all your data by default. If instead you'd like Docker to automatically clean up the container and remove the file system when the container exits, you can add the –rm flag:

```
docker run -it --rm hellerd/sshmm
```

4. To access data located on your machine (e.g. in `/home/someuser/data`) from inside the container, use the `-v` option:

```
docker run -it -v /home/someuser/data:/data hellerd/sshmm
```

This will make your data directory (`/home/someuser/data`) available inside the container in the `/data` directory. You can now run ssHMM on this data, e.g. `train_seqstructhmm /data/sequences.fasta /data/shapes.txt`.

3.1.3 Installation as a Python package

Prerequisites

- GHMM (http://ghmm.org/)
- GraphViz (http://www.graphviz.org/)

For preprocessing BED files with preprocess_dataset (tested only on Linux and macOS):

- bedtools (https://github.com/arq5x/bedtools2)
- RNAshapes 2.1.6 (https://bibiserv.cebitec.uni-bielefeld.de/download/tools/rnashapes.html)
- RNAstructure 5.8.1 (http://rna.urmc.rochester.edu/rnastructure.html)

Installation of prerequisites

1. Install prerequisites for GHMM as described on http://ghmm.sourceforge.net/installation.html. The commands for Ubuntu are:

```
sudo apt-get update
sudo apt-get install build-essential automake autoconf libtool
sudo apt-get install python-dev
sudo apt-get install libxml++2.6-dev
sudo apt-get install swig
```

2. Download and unpack GHMM from https://sourceforge.net/projects/ghmm/
3. Install GHMM as described on http://ghmm.sourceforge.net/installation.html. The commands for Ubuntu are:

```
cd ghmm
sh autogen.sh
sudo ./configure
sudo make
sudo make install
sudo ldconfig
```

4. Install GraphViz. On Ubuntu:

```
sudo apt-get install graphviz
sudo apt-get install libgraphviz-dev
```

5. Install pip if not already installed. On Ubuntu:

```
sudo apt-get install python-pip
```

6. Install PyGraphViz:

```
sudo PKG_CONFIG_ALLOW_SYSTEM_LIBS=OHYESPLEASE pip install pygraphviz
```

7. Install bedtools as described on http://bedtools.readthedocs.io/en/latest/content/installation.html
8. Download and install RNAshapes from https://bibiserv.cebitec.uni-bielefeld.de/download/tools/rnashapes.html.
9. Download and install RNAstructure from http://rna.urmc.rochester.edu/register.html.

Installation of ssHMM

1. Download ssHMM from https://github.molgen.mpg.de/heller/ssHMM
2. Install ssHMM:

```
#as root
sudo python setup.py install

#as non-root
python setup.py install --user
```

This will install ssHMM and the following python package dependencies: numpy, graphviz, pygraphviz, weblogo, future, logging_exceptions, forgi. If setuptools fails to install any of the dependencies, try to install it separately (e.g. with `sudo pip install numpy`).

We distribute ssHMM both as a Python package and as a Docker image (similar to a virtual machine). There are three ways to install ssHMM:

Conda installation:

- quick and easy installation
- requires Conda
- recommended for most users

Docker installation:

- quick and easy installation
- requires Docker
- requires sudo rights
- all dependencies of ssHMM are pre-installed in the image
- recommended for Linux beginners

Manual installation:

- more difficult installation
- dependencies of ssHMM have to be installed manually
- recommended only for experienced users

3.2 Tutorial

This tutorial will make you familiar with ssHMM and explain its usage.

3.2.1 0. Installation

To begin this tutorial, first install ssHMM as described in the *Installation* section. To make sure that the installation was successfull, check whether you can execute the ssHMM scripts:

```
preprocess_dataset -h
train_seqstructhmm -h
batch_seqstructhmm -h
```

If all goes well, you should see the help messages of the ssHMM scripts.

3.2.2 1. Download a CLIP-Seq dataset

Now we need a CLIP-Seq dataset to work with. On Github we provide a repository of 25 CLIP-Seq and 24 synthetic datasets (https://github.molgen.mpg.de/heller/ssHMM_data). For this tutorial, we use the PUM2 CLIP-Seq dataset and download the pre-processed files for sequence and structure:

```
cd /home/myuser
mkdir clipseq
cd clipseq
wget https://github.molgen.mpg.de/raw/heller/ssHMM_data/master/clip-seq/fasta/PUM2/
↪positive.fasta
wget https://github.molgen.mpg.de/raw/heller/ssHMM_data/master/clip-seq/shapes/PUM2/
↪positive.txt
```

3.2.3 2. Start Docker image (if installed with Docker)

If ssHMM is installed as a Docker image, you first have to start the image:

```
docker run -t -i -v /home/myuser/:/home/myuser/ hellerd/sshmm
```

This boots the ssHMM image and opens a command line to control the running container. The `-v` option makes the home directory (containing the `clipseq` directory) available from within the container. Continue with the tutorial by running all commands in the container.

3.2.4 3. Inspect the dataset

Let's have a look at the two files we downloaded:

```
head /home/myuser/clipseq/positive.fasta
```

```
>chr6:89794035-89794147(+)
aaaaaattacatacaaacagCTTGTATTATATTTTATATTTTGTAAATACTGTATACCATGTATTATGTGTATATTGTTCATACTTGAGAGGtatattata
>chr10:102767488-102767578(+)
cacccaggtttatggcctcgTTTTCACTTGTATATTTTTCACACTGTAAATTTCTTGTACAAACCCAAAGaaaaaattaaaaaaaatttt
>chr2:99234790-99234904(+)
taactgtgtcaacagtattgTGAAGTGATCATTTCTTGTAAAACTTGTAAATAAACTATCATCTTTGTAGATATCTTAAAGGTGTAAAGTTTGCaaattt
>chr12:49521563-49521638(-)
gtgatcatgtcttttccatgTGTACCTGTAATATTTTTCCATCATATCTCAAAGTaaagtcattaacatcaaaag
```

The FASTA file contains the nucleotide sequence of the PUM2 binding sites as determined by a CLIP-Seq experiment. Every two lines of the FASTA file hold one binding site. The first line (beginning with >) specifies the genomic location of the site while the second line contains the genomic sequence.

```
head /home/myuser/clipseq/positive.txt
```

```
>chr6:89794035-89794147(+)
EEEEEEEEESSSSSSSSIISSISSSSISSSSSSSSIIISSIISSSIIISSISSSSSSSSSSHHSSSSISSSSSSISSIIISSSIISSSSSSSSSSSISSSS
 ↪0.008824
>chr10:102767488-102767578(+)
EEEEESSSSHHHHHSSSSMMMMMMMMMSSSSSSSHHHHHHHHHHHHHHHHHHHHHHSSSSSSSEEEEEEEEEEEEEEEEEEEEEEEEEEE␣
 ↪0.0312072
EEEEESSSSHHHHHSSSSMMMMMMMMMSSSSSSSHHHHHHHHHHHHHHHHHHHHHHSSSSSSSMMMMMMMMMMSSSSSSHHHHHHSSSSSS␣
 ↪0.0163077
>chr2:99234790-99234904(+)
EESSSSSHHHHSSSSSMMMMMMMMMMMMMMSSSSSSSIIISSSISSSSSSSSIIIIIIISSSSSSSSISSHHHHSSSSSSSSSSIIIISSSSSSSSISSSI
 ↪0.0677326
EESSSSSHHHHSSSSSMMMSSSSHHHHHSSSSSSSSSIIISSSISSSSSSSSIIIIIIISSSSSSSSISSHHHHSSSSSSSSSSIIIISSSSSSSSISSSI
 ↪0.0031042
>chr12:49521563-49521638(-)
SSSSSIISSISSSSIIISSSSSHHHHHHHHHHHHHHHHHHHHSSSSSIIISSSSSSIISSSSSEEEEEEEEEEEE 0.098404
```

The structure file contains the predicted secondary structures of the binding sites. The prediction were performed with the `RNAshapes` tool. Again, the lines starting with > specify the genomic location of a binding site. The subsequent lines contain the predicted structural context of each nucleotide in the FASTA file. Note that these structure sequences have the same length as the nucleotide sequences from the FASTA file we have looked at before.

3.2.5 4. Training ssHMM on the dataset

Now we can train ssHMM on the CLIP-Seq dataset we downloaded:

```
cd /home/myuser
mkdir results
train_seqstructhmm clipseq/positive.fasta clipseq/positive.txt -o results/
```

This creates a new directory `results` and starts the training of ssHMM using the `train_seqstructhmm` script. `train_seqstructhmm` has two mandatory arguments: the sequence and the structure file. We use the files that we downloaded and additionally tell ssHMM to write its output into the new `results` directory. For a description of all arguments of `train_seqstructhmm` see its *reference*.

While the `train_seqstructhmm` script runs, it writes information to the standard output. For more information on the output, refer to the *Output* section. When the script finishes, it prints messages on standard output that look similar to:

```
2017-02-23 11:45:33,381 - main_logger - INFO - Terminate model after 7000 iterations.
2017-02-23 11:45:33,381 - main_logger - INFO - Completed training. Write sequence␣
 ↪logos..
2017-02-23 11:45:35,675 - main_logger - INFO - Completed writing sequence logos.␣
 ↪Print model graph..
2017-02-23 11:45:35,987 - main_logger - INFO - Printed model graph: ./job_170223_
 ↪114151/final_graph.png. Write model file..
2017-02-23 11:45:35,992 - main_logger - INFO - Wrote model file: ./job_170223_114151/
 ↪final_model.xml
2017-02-23 11:45:35,992 - main_logger - INFO - Finished ssHMM successfully.
```

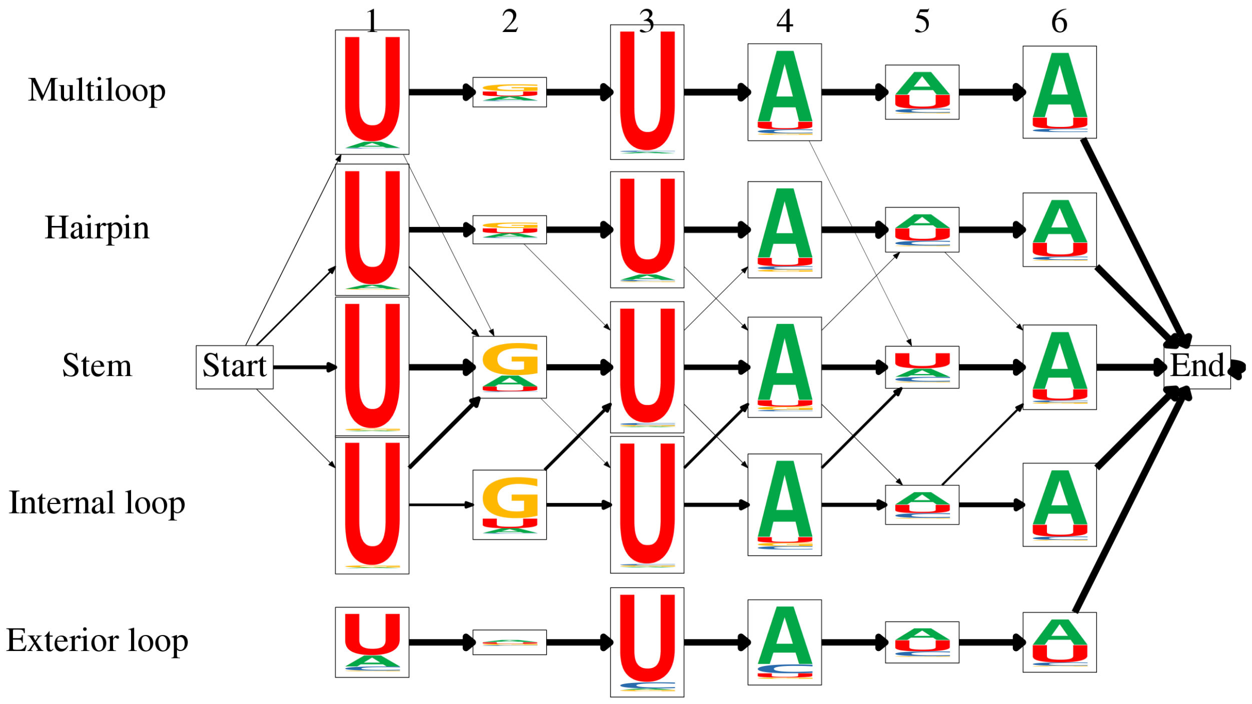
The lines tell you how many iterations the training took (7000) and where you can find a graph and an XML of the trained model.

3.2.6 5. Inspect the trained model

Hint: If you ran ssHMM in the Docker container, it is now time to exit from the container. As the `results` directory is a subdirectory of `/home/myuser`, the training results can also be found on your host machine. Exiting from the Docker container is easy:

```
exit
```

We can now have a look at the model graph. See *Training output* for an explanation of what the model graph shows.



Congratulations, you finished our tutorial! Check out the *Reference* section for more information about the ssHMM scripts.

3.3 Output

3.3.1 Training output

The *train_seqstructhmm* script writes all its output into the given output directory (by default: the current directory, configurable with the `--output_directory` parameter). There, it creates a job directory with the name `<job_name>_<date>_<time>`. During the runtime of the script, the following files are written:

Logs:

- `<job_name>_verbose.log`
- `<job_name>_numbers.log`

Model graphs:

- `graphs/graph_<iteration>.png`

- `final_graph.png`

Model files:

- `models/model_<iteration>.xml`
- `final_model.xml`

Sequence logos:

- `logo_global.png`
- `logo_global.txt`
- `logo_hairpin.png`
- `logo_hairpin.txt`
- `logo_best_sequences.png`
- `logo_best_sequences.txt`

Logs

The verbose log `<job_name>_verbose.log` is identical to the standard output of the script. Each line contains the current time and the logging level. At the top, the full command that called the script and the chosen options are printed. In every training iteration, the log contains the current loglikelihoods of the input data given the model and the current information content of the model in six different variants. The information content is either obtained from the top-scoring 1000 sequences or computed directly from the model. Additionally, we distinguish sequence, structure, and sequence-structure information content. At the end, the log contains the number of training iterations and the locations of the final model graph and model file.

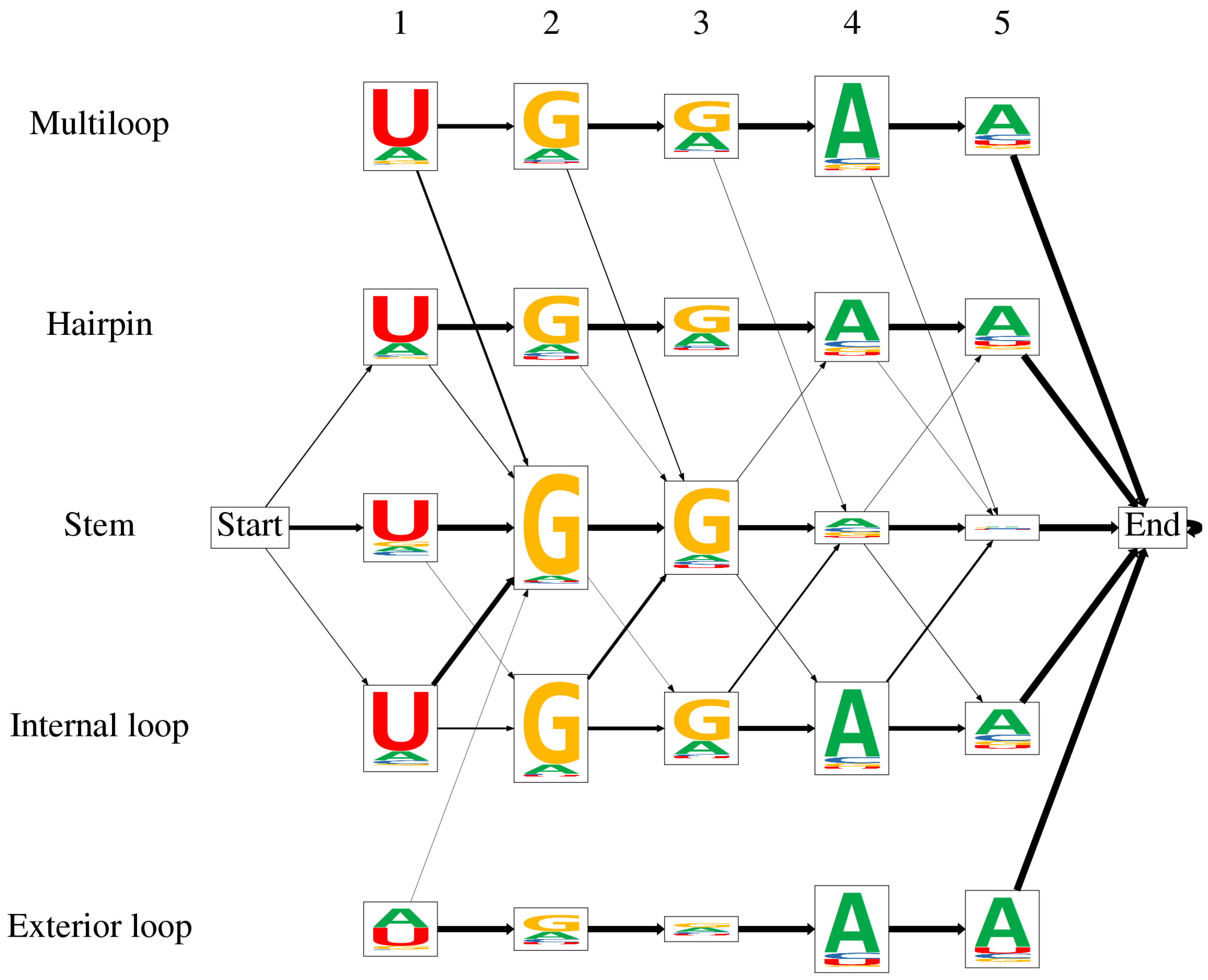
The numbers log `<job_name>_numbers.log` contains the same information for each iteration as the verbose log but in a condensed, comma-separated format. Each line holds information for one iteration. The columns are:

- Iteration number
- Sequence loglikelihood
- Sequence-structure loglikelihood
- Average per-position information content of sequence (from top 1000 sequences)
- Average per-position information content of structure (from top 1000 sequences)
- Average per-position information content of sequence and structure (from top 1000 sequences)
- Average per-position information content of sequence (from model)
- Average per-position information content of structure (from model)
- Average per-position information content of sequence and structure (from model)

Model graphs

The graph images provide a clear and intuitive visualization of the motif that has been found by training the model. HMMs are graphical models that lend themselves to a visualization as a graph. The states and emissions of an HMM can be depicted as nodes while the transition and emission probabilities can be represented as weighted edges between the nodes.

As an example, the following image shows the output of ssHMM after it has been trained on a CLIP-Seq dataset of the RBP DGCR8. The states of the ssHMM, including the start and the end state, are represented by rectangular boxes that are arranged into rows and columns. The columns represent the motif positions while the rows represent the five structural contexts of RNA.



The emissions of the ssHMM are the four nucleotides A, C, G, and T. Sequence logos are used to visualize the emission probabilities of each state directly in the state's box. The heights of the nucleotide letters in the sequence logos depend on two properties: Firstly, the relative height of a letter reflects its prevalence in this state, i.e. its emission probability. Secondly, the total height of the stack of four nucleotides is scaled according to the information content of the emission probabilities in this state. Consequently, states with a high information content stand out because of their size while those with more uniform emission probabilities are smaller. Different colors for the four nucleotides make it very easy to immediately grasp the sequence motif defined by the ssHMM. For this concrete RBP, the sequence motif is UGGAA.

The transition probabilities between the states are visualized as arrows. The thicker an arrow between two states, the more likely is a transition between the two. The most important transitions originate from the start state because they determine in which structural context the motif starts. Often, the motif remains in one particular context for its entire length (as can be seen for the hairpin and stem context in the graph above} but not for the internal loop context). To reduce clutter and increase clarity, unlikely transitions with a probability of less than 0.05 are completely omitted. This can be observed, for example, between the start state and both multiloop and exterior loop contexts. The RBP depicted in the graph above preferably binds stem regions of RNA but to a lesser extent also hairpin loops and internal loops.

By default, the training script outputs a graph image of the current state of the model every `i` iterations (as defined by the `--termination_interval` argument) into the `graphs/` subdirectory. After the training is completed, the final model graph is written into the file `final_graph.png`.

Model files

Every `i` iterations, the model in its current state is exported as an XML file into `models/model_<iteration>.xml`. After the training is completed, the final model is exported as `final_model.xml`. The exported XML files hold all model parameters (e.g. states, emissions, transition probabilities, emission probabilities and initial probabilities). Advanced users can use the `ghmm` Python package to import and use the exported HMMs further.

Sequence logos

Beside the comprehensive visualization of the model as a model graph, ssHMM supports the visualization of the trained motif as simple sequence logos. Three different sequence logos are produced after completion of the training:

- a global sequence logo `logo_global.png` which shows the average sequence motif over all structural contexts
- a hairpin sequence logo `logo_hairpin.png` which shows the sequence motif in the hairpin context
- a sequence logo `logo_best_sequences.txt` which is generated from the 1000 sequences that are most likely under the model

All three sequence logos are exported as PNG image and text file. The text files have the following format: The first line lists the emissions (in most cases, A, C, G and T). Each subsequent line contains one motif position with the tab-separated probabilities of the respective emissions.

3.3.2 Batch-Training output

The *batch_seqstructhmm* script writes all its output into the given batch directory (set with the `batch_directory` parameter). There, it creates a batch job directory with the name `batch_<date>_<time>`. In the batch job directory, `batch.log` holds the standard output of the script. Additionally, it contains one directory for each protein in the batch: `<protein>_<struct. type>_ml<motif len.>_fl<flex.>_bs<block size>_ti<term. interval>_tt<threshold>`

These directories have the same structure as other training output directories. See *Training output* for details.

During its execution, ssHMM outputs helpful information both on standard output and into the given output directory. For the output of the *train_seqstructhmm* script, see *Training output*. For the output of the *batch_seqstructhmm* script, see *Batch-Training output*.

3.4 Preprocessing

This tutorial will lead you through the preprocessing of your own CLIP-Seq dataset for subsequent analysis with ssHMM. All you need is a file in BED format (https://genome.ucsc.edu/FAQ/FAQformat.html#format1). This file contains genomic regions bound by a specific RBP as determined in a CLIP-Seq experiment. Theoretically however, you can use any BED file with RNA regions that you want to find the sequence-structure motif for.

3.4.1 1. Preparation

For the preprocessing, you need the reference genome on which the genome regions in the BED file are defined. If you performed alignment and peak calling e.g. on hg19, this same reference genome has to be used in the preprocessing now. You require two files for the genome:

- the genome sequence in FASTA format
- the chromosome sizes of the genome (e.g. from http://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/hg19.chrom.sizes)

3.4.2 2. Run the preprocessing script

Now, we can start the preprocessing. You need to pass the following parameters:

- a working directory for the output
- a dataset name
- the input BED file
- the genome sequence file
- the chromosome sizes file

```
preprocess_dataset WORKING_DIR DATASET_NAME INPUT_BED GENOME_SEQ GENOME_SIZES
```

Additionally, you could set optional parameters (see `preprocess_dataset --help`).

The script will now perform the following steps:

1. Filter BED file
2. Elongate BED file for later structure prediction
3. Fetch genomic sequences for elongated BED file
4. Produce FASTA file with genomic sequences in viewpoint format
5. Secondary structure prediction with RNAshapes
6. Secondary structure prediction with RNAstructures

3.4.3 3. Results

Once, the script has finished, we can inspect the results:

- `WORKING_DIR/fasta/DATASET_NAME/positive.fasta` - genomic sequences in viewpoint format
- `WORKING_DIR/shapes/DATASET_NAME/positive.txt` - secondary structures of genomic sequence (predicted by RNAshapes)
- `WORKING_DIR/structures/DATASET_NAME/positive.txt` - secondary structures of genomic sequence (predicted by RNAstructures)

You can now proceed to analyze these genomic sequences and secondary structures for a common sequence-structure motif with the `train_seqstructhmm` script (see *4. Training ssHMM on the dataset*).

3.5 Reference

3.5.1 preprocess_dataset

Pipeline for the preparation of a CLIP-Seq dataset in BED format. The pipeline consists of the following steps: 1 - Filter BED file 2 - Elongate BED file for later structure prediction 3 - Fetch genomic sequences for elongated BED file 4 - Produce FASTA file with genomic sequences in viewpoint format 5 - Secondary structure prediction with RNAshapes 6 - Secondary structure prediction with RNAstructures

DEPENDENCIES: This script requires bedtools (shuffle, slop, getfasta), RNAshapes, and RNAstructures.

A working directory and a dataset name (e.g., the protein name) have to be given. The output files can be found in: - <workingdir>/fasta/<dataset_name>/positive.fasta - genomic sequences in viewpoint format - <workingdir>/shapes/<dataset_name>/positive.txt - secondary structures of genomic sequence (predicted by RNAshapes) - <workingdir>/structures/<dataset_name>/positive.txt - secondary structures of genomic sequence (predicted by RNAstructures)

For classification, a negative set of binding sites with shuffled coordinates can be generated with the –generate_negative option. For this option, gene boundaries are required and need to be given as –genome_genes. They can be downloaded e.g. from the UCSC table browser (http://genome.ucsc.edu/cgi-bin/hgTables). Choose the most recent GENCODE track (currently GENCODE Gene V24lift37->Basic (for hg19) and All GENCODE V24->Basic (for hg38)) and 'BED' as output format.

```
usage: preprocess_dataset [-h] [--disable_filtering] [--disable_RNAshapes]
                          [--disable_RNAstructure] [--generate_negative]
                          [--min_score MIN_SCORE] [--min_length MIN_LENGTH]
                          [--max_length MAX_LENGTH] [--elongation ELONGATION]
                          [--genome_genes GENOME_GENES] [--skip_check]
                          working_dir dataset_name input genome genome_sizes
```

Positional Arguments

working_dir	working/output directory
dataset_name	dataset name
input	input file in .bed format
genome	reference genome in FASTA format
genome_sizes	chromosome sizes of reference genome (e.g. from http://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/hg19.chrom.sizes)

Named Arguments

--disable_filtering, -f	skip the filtering step Default: False
--disable_RNAshapes	skip secondary structure prediction with RNAshapes Default: False
--disable_RNAstructure	skip secondary structure prediction with RNAstructures Default: False
--generate_negative, -n	generate a negative set for classification Default: False
--min_score	filtering: minimum score for binding site (default: 0.0) Default: 0.0
--min_length	filtering: minimum binding site length (default: 8) Default: 8

--max_length	filtering: maximum binding site length (default: 75) Default: 75
--elongation	elongation: span for up- and downstream elongation of binding sites (default: 20) Default: 20
--genome_genes	negative set generation: gene boundaries
--skip_check, -s	skip check for installed prerequisites Default: False

3.5.2 train_seqstructhmm

Trains a Hidden Markov Model for the sequence-structure binding preferences of an RNA-binding protein. The model is trained on sequences and structures from a CLIP-seq experiment given in two FASTA-like files. During the training process, statistics about the model are printed on stdout. In every iteration, the current model and a visualization of the model can be stored in the output directory. The training process terminates when no significant progress has been made for three iterations.

```
usage: train_seqstructhmm [-h] [--motif_length MOTIF_LENGTH] [--random]
                          [--flexibility FLEXIBILITY]
                          [--block_size BLOCK_SIZE] [--threshold THRESHOLD]
                          [--job_name JOB_NAME]
                          [--output_directory OUTPUT_DIRECTORY]
                          [--termination_interval TERMINATION_INTERVAL]
                          [--no_model_state] [--only_best_shape]
                          training_sequences training_structures
```

Positional Arguments

training_sequences	FASTA file with sequences for training
training_structures	FASTA file with RNA structures for training

Named Arguments

--motif_length, -n	length of the motif that shall be found (default: 6) Default: 6
--random, -r	Initialize the model randomly (default: initialize with Baum-Welch optimized sequence motif) Default: False
--flexibility, -f	greedyness of Gibbs sampler: model parameters are sampled from among the top f configurations (default: f=10), set f to 0 in order to include all possible configurations Default: 10
--block_size, -s	number of sequences to be held-out in each iteration (default: 1) Default: 1

--threshold, -t	the iterative algorithm is terminated if this reduction in sequence structure log-likelihood is not reached for any of the 3 last measurements (default: 10) Default: 10.0
--job_name, -j	name of the job (default: “job”) Default: “job”
--output_directory, -o	directory to write output files to (default: current directory) Default: “.”
--termination_interval, -i	produce output every <i> iterations (default: i=100) Default: 100
--no_model_state, -w	do not write model state every i iterations Default: False
--only_best_shape	train only using best structure for each sequence (default: use all structures) Default: False

3.5.3 batch_seqstructhmm

Trains multiple Hidden Markov Models for the sequence-structure binding preferences of a given set of RNA-binding proteins. The models are trained on sequences and structures in FASTA format located in a given data directory. During the training process, statistics about the models are printed on stdout. In every iteration, the current model and a visualization of the model are stored in the batch directory. The training processes terminate when no significant progress has been made for three iterations.

```
usage: batch_seqstructhmm [-h] [--cores CORES]
                          [--structure_type STRUCTURE_TYPE]
                          [--motif_length MOTIF_LENGTH] [--baum_welch]
                          [--flexibility FLEXIBILITY]
                          [--block_size BLOCK_SIZE] [--threshold THRESHOLD]
                          [--termination_interval TERMINATION_INTERVAL]
                          data_directory proteins batch_directory
```

Positional Arguments

data_directory	data directory; must contain the sequence files under fasta/<protein>/positive.fasta and structure files under <structure_type>/<protein>/positive.txt
proteins	list of RNA-binding proteins to analyze (surrounded by quotation marks, separated by whitespace)
batch_directory	directory for batch output

Named Arguments

--cores, -c	number of cores to use (if not given, all cores are used)
--structure_type, -s	structure type to use; must match location of structure files (see data_directory argument above) (default: shapes) Default: “shapes”

--motif_length, -n	length of the motifs that shall be found (default: 6) Default: 6
--baum_welch, -b	should the models be initialized with a Baum-Welch optimized sequence motif (default: yes) Default: True
--flexibility, -f	greedyness of Gibbs sampler: model parameters are sampled from among the top f configurations (default: f=10), set f to 0 in order to include all possible configurations Default: 10
--block_size	number of sequences to be held-out in each iteration (default: 1) Default: 1
--threshold, -t	the iterative algorithm is terminated if this reduction in sequence structure log-likelihood is not reached for any of the 3 last measurements (default: 10) Default: 10.0
--termination_interval, -i	produce output every <i> iterations (default: i=100) Default: 100