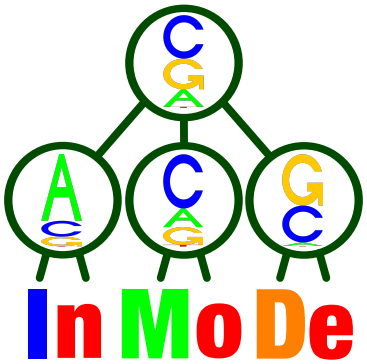

version 1.0

User guide

Ralf Eggeling, Ivo Grosse, Jan Grau

If you use InMoDe, please cite:
R. Eggeling, I. Grosse, and J. Grau. InMoDe: tools for learning and visualizing intra-motif dependencies of DNA binding sites. *Bioinformatics*, 2016. doi: 10.1093/bioinformatics/btw689

1 Overview

InMoDe is a collection of seven tools for learning, leveraging, and visualizing **in**tra-**mo**tif **de**pendencies within DNA binding sites and similar functional nucleotide sequences.

Four tools perform different variants of learning, i.e., determining the structure and parameters of inhomogeneous parsimonious Markov models (iPMMs) [1] based on given input data:

- `SimpleMoDe` learns a single iPMM from a set of gaplessly pre-aligned DNA sequences of same length.
- `DeNovoMoDe` performs an iPMM-based de novo motif discovery within a set of longer DNA sequences such as ChIP-seq positive fragments.
- `MixtureMoDe` learns a mixture of several iPMMs from a set of gaplessly pre-aligned DNA sequences of same length.
- `FlexibleMoDe` allows to further customize the learning task while still containing all three previous tools as special cases.

The remaining three tools perform different applications of previously learned iPMMs:

- `ScanApp` searches in DNA sequences for occurrences of a given iPMM.
- `ClassificationApp` performs a binary classification of sequences that can be scored by iPMMs.
- `VisualizationApp` allows a customized visualization of model structure and parameters of an iPMM in terms of a conditional sequence logo.

Detailed descriptions of these tools are available in Chapter 5.

InMoDe provides three user-interfaces:

- `InMoDeGUI` is a JavaFX-based GUI interface for most intuitive use.
- `InMoDeCLI` is a command-line interface for running InMoDe e.g. on a cluster.
- `InMoDeGalaxy` allows to integrate InMoDe into Galaxy pipelines.

Detailed descriptions of the three interfaces are available in Chapter 3.

2 Download and installation

The three variants of InMoDe can be downloaded from http://www.jstacs.de/index.php/InMoDe. They are available as runnable .jar files

- InMoDeGUI.jar
- InMoDeCLI.jar
- InMoDeGalaxy.jar

and run on Linux, Mac OS X, and Windows, but require an existent Java installation (8u74 or later). In addition, there are two user-friendly alternatives for installing InMoDeGUI, namely (i) a DMG for installation under Mac OS X, and (ii) a Windows installer. Both do not require a recent Java, as they automatically install the required libraries to the local machine.

2.1 Installation from the OS X DMG file

The OS X DMG file contains the InMoDe application (Figure 2.1, left), which may be copied to the Applications folder or any other location that the user considers appropriate. Afterwards, InMoDe can be started by double-clicking on the InMoDe application.

2.2 Windows installer

The windows installer contains an appropriate version of Java in addition to InMoDe itself. This version of Java is installed together with InMoDe and should not interfere with another Java version already installed on the computer. After downloading the windows installer, the installation process can be started (Figure 2.1, right). After the installation has finished, InMoDe can be found in the list of Apps and a shortcut is created on the Desktop.

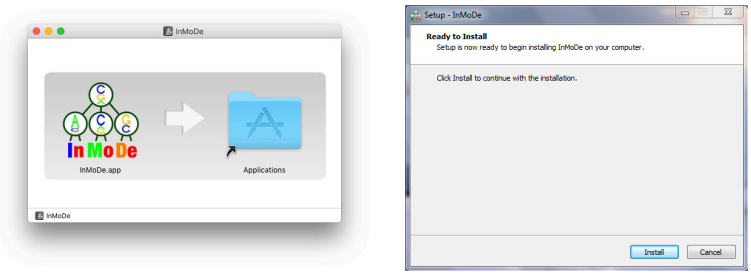

Figure 2.1: Installation of InMoDe for Max OS X (left) and Windows (right).

3 User interfaces

InMoDe provides three different user interfaces that are described below.

3.1 Graphical user interface

The graphical user interface (GUI) allows us to run InMoDe in an interactive mode locally on a desktop computer. It is sufficient to explore a tool's capabilities and to perform single experiments. For larger studies involving more than a few data sets, it is recommended to use InMoDeCLI or InMoDeGalaxy.

The GUI can be started by calling

```
java -jar InMoDeGUI.jar
```

or by double-clicking on the .jar-file. For user's convenience, we also provide packages for Mac OS and Windows systems that come pre-packed with Java (see Chapter 2). The main window is divided into three frames (Figure 3.1).

The left frame (Figure 3.1A) allows selecting the desired tool, specifying the tool parameters, and starting jobs. Buttons labeled by questions marks open a popup-help for the corresponding tool, which contains a tool-internal summary of the information from Chapter 5 of this user guide.

The frame in the bottom-right corner (Figure 3.1B) shows by default a list with (i) all data sets that have been loaded into the workspace and (ii) all results that have been produced as tool output. It is possible to rename each data set and result. The rename-mode is activated by clicking on a name, and renaming is confirmed by hitting ENTER. The save-button next to each item allows to store the corresponding item on disk. For list-results such as "Learned iPMM(3)" in Figure 3.1, which summarize the total output of one job, all individual results are saved at once. The remove-button deletes a dataset or a result. The restore-button (List-results only) restores the parameter values by which the result has been produced at the parameter selection in the left frame. Alternative to the data view, the bottom-right frame can also display a protocol as shown in Figure 3.1E. It contains information about the progress of a tool and a few statistics about the final result after a job has finished. The content of a protocol can be saved to disk and erased using the corresponding buttons. Both views can be switched by the tab at the top of the frame.

The frame in the top-right corner can display images and text files. It can be used to show (conditional) sequence logos as shown in Figure 3.1C, text results such as precise learned parameter values as shown in Figure 3.1F, and the content of loaded data sets. When text is displayed, arbitrary lines can be marked and copied to clipboard by STRG+C or an equivalent command.

The bar at the bottom of the screen (Figure 3.1D) contains a progress bar that indicates the state of the current tool and shows how many jobs are currently pending in the queue, the list of which can be inspected by clicking on button "Tasklist". In addition, the current workspace can be saved and another workspace can be loaded from disk by using the corresponding buttons on the right side of the bar.

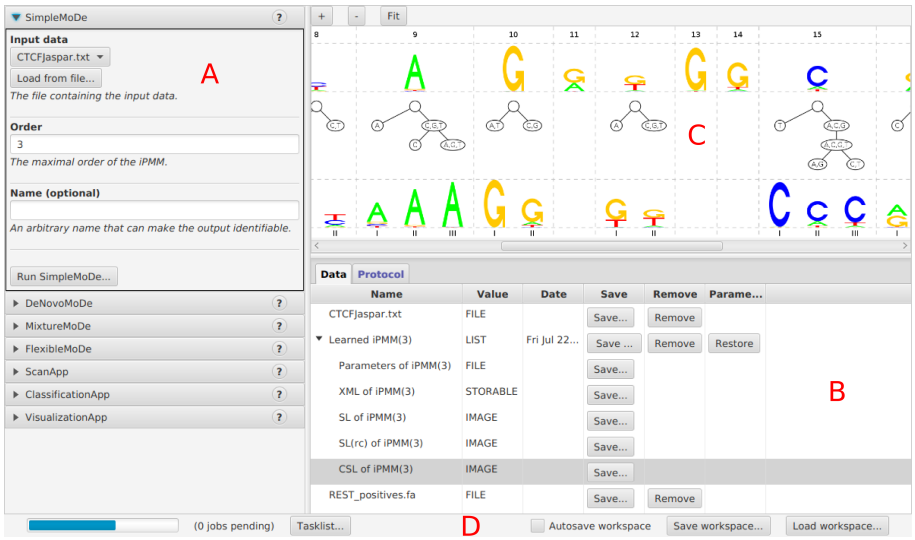

Figure 3.1: A: Tool and parameter selection. B: Data and result list. C: File content box with conditional sequence logo plot displayed. D: Status bar

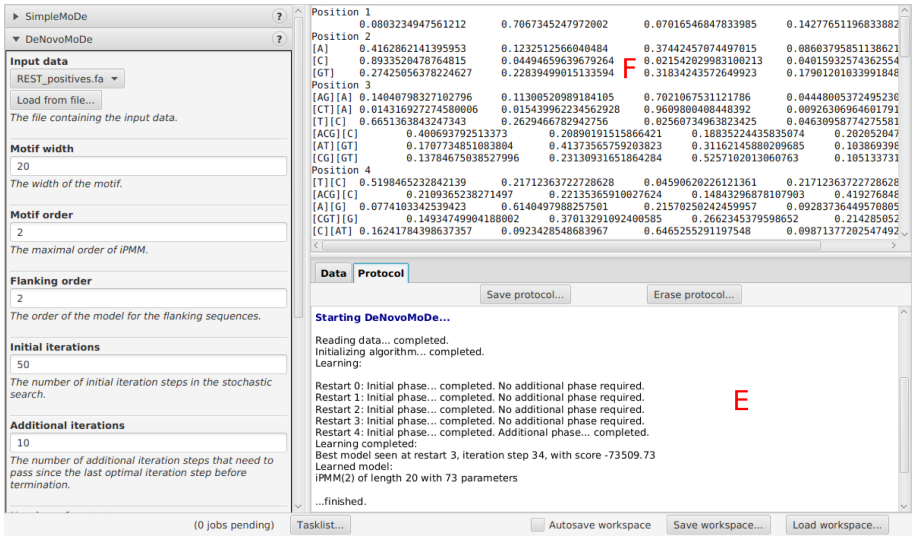

Figure 3.2: E: Protocol display. F: File content box with a text result displayed.

3.2 Command line interface

The command line interface is provided by `InMoDeCLI.jar`. A list of all tools contained in InMoDe is obtained by calling

```
java -jar InMoDeCLI.jar
```

which yields

```
Available tools:

        simple - SimpleMoDe
        denovo - DeNovoMoDe
        mixture - MixtureMoDe
        flexible - FlexibleMoDe
        scan - ScanApp
        classify - ClassificationApp
        visualize - VisualizationApp

Syntax: java -jar InMoDeCLI.jar <toolname> [<parameter=value> ...]

Further info about the tools is given with
        java -jar InMoDeCLI.jar <toolname> info

Tool parameters are listed with
        java -jar InMoDeCLI.jar <toolname>
```

Calling InMoDeCLI.jar with the corresponding tool name, e.g.,

```
java -jar InMoDeCLI.jar simple
```

yields a list of input parameters:

```
At least one parameter has not been set (correctly):

Parameters of tool "SimpleMoDe" (simple, version: 1.0):
i - Input data (The file containing the input data.)    = null
o - Order (The maximal order of the iPMM., valid range = [0, 7], default = 2)    = 2
n - Name (An arbitrary name that can make the output identifiable., OPTIONAL)    = null
outdir - The output directory, defaults to the current working directory (.)    = .
```

The parameter set incorrectly here is Input data, which is, unlike Name, not optional. The command

```
java -jar InMoDeCLI.jar simple i=test.txt
```

triggers the tool to learn an second-order iPMM on the file `test.txt`, and writes the output into the current directory, whereas the command

```
java -jar InMoDeCLI.jar simple o=4 i=test.txt outdir=myResults
```

triggers the tool to learn a fourth-order iPMM on the same file and writes the output into directory `myResults`.

3.3 Galaxy integration

In additional to the GUI and the command line interface, InMoDe can also used within Galaxy pipelines.

3.3.1 Web-server

A server with all tools of InMoDe is available for public use at http://galaxy.informatik.uni-halle.de. The provided web-server puts a certain limit on the complexity of runnable jobs for the learning tools, such as

- Order of iPMM(s) ≤ 3
- Number of mixture components ≤ 5
- Initial iterations ≤ 100,
- Additional iterations ≤ 20,
- Number of restarts ≤ 5

For unlimited use, it is recommended to use InMoDeCLI or to integrate `InMoDeGalaxy.jar` in an own Galaxy instance.

3.3.2 Local integration

The installation of InMoDe in an own Galaxy installation just requires a few simple steps:

1. Downloading the InMoDeGalaxy.jar file from the project website.
2. Copying the InMoDeGalaxy.jar file to a directory that can be found by Galaxy. In the following, we assume that this is a directory "InMoDe" in the Galaxy "tools" directory.
3. `cd` to that directory and issuing

   ```
   java -jar InMoDeGalaxy.jar --create
   ```

   which creates XML tool descriptions for all individual InMoDe tools.
4. Adding these tools to the Galaxy tool config file, which typically is "config/tool_conf.xml". For instance, the corresponding entries may read

   ```
   <tool file="InMoDe/simple.xml" />
   <tool file="InMoDe/denovo.xml" />
   <tool file="InMoDe/mixture.xml" />
   <tool file="InMoDe/flexible.xml" />
   <tool file="InMoDe/scan.xml" />
   <tool file="InMoDe/classify.xml" />
   <tool file="InMoDe/visualize.xml" />
   ```

   which may be enclosed by an additional `<section>` tag.
5. Making Galaxy aware of the added tools, e.g., by restarting the Galaxy instance. Now, the tools should appear in the Galaxy tools bar. For executing InMoDe tools, Java must be installed on the complete Galaxy system (e.g. including compute nodes when being run on a cluster).

4 General principles

There are a few general principles that apply, if not stated otherwise, to all tools within InMoDe.

4.1 Input data

All input data (except the optional data weights for `FlexibleMoDe`) are expected to consist of DNA sequences and to satisfy the following conventions:

If the first symbol in the input file equals >, the file is interpreted as .fasta file, where individual sequences are separated by comment-lines starting with >. If the first symbol does not equal >, the file is interpreted as plain text file, where each line break indicates the start of a new sequence.

All tools of InMoDe work on ungapped DNA sequences over the (case-insensitive) four-letter DNA alphabet. If sequences over the extended IUPAC alphabet including ambiguous symbols are used as input, each ambiguous symbol is replaced by a nucleotide randomly sampled from the nucleotide distribution of the remaining data set. This random replacement is being performed every time anew once a particular data set is used, so the results may and will differ from run to run due to the random nature of the replacement procedure. If a more systematic handling of ambiguous nucleotides is desired, the user should pre-process the data manually.

To be precise...

The replacement procedure works as follows. Let $\mathcal{A} = \{\mathsf{A},\mathsf{C},\mathsf{G},\mathsf{T}\}$ denote the standard DNA alphabet and $\mathcal{A}' = \{\mathsf{R},\mathsf{Y},\mathsf{S},\mathsf{W},\mathsf{K},\mathsf{B},\mathsf{D},\mathsf{H},\mathsf{V},\mathsf{N}\}$ the ambigous symbols in the extended IUPAC alphabet $\mathcal{A} \cup \mathcal{A}'$. First compute for all $a \in \mathcal{A}$

$$f_a = \frac{N_a}{\sum_{a' \in \mathcal{A}} N_{a'}} \tag{4.1}$$

where N_a denotes the absolute frequencies of symbol a in the data set. Next, replace each symbol $b \in \mathcal{A}'$ encountered in the data by a randomly chosen symbol from $\mathcal{A}$ according to the following normalized distributions:

IUPAC symbol		distribution (unnormalized)
R	$\sim$	$(f_\mathsf{A}, 0, f_\mathsf{G}, 0)$
Y	$\sim$	$(0, f_\mathsf{C}, 0, f_\mathsf{T})$
S	$\sim$	$(0, f_\mathsf{C}, f_\mathsf{G}, 0)$
W	$\sim$	$(f_\mathsf{A}, 0, 0, f_\mathsf{T})$
K	$\sim$	$(0, 0, f_\mathsf{G}, f_\mathsf{T})$
M	$\sim$	$(f_\mathsf{A}, f_\mathsf{C}, 0, 0)$
B	$\sim$	$(0, f_\mathsf{C}, f_\mathsf{G}, f_\mathsf{T})$
D	$\sim$	$(f_\mathsf{A}, 0, f_\mathsf{G}, f_\mathsf{T})$
H	$\sim$	$(f_\mathsf{A}, f_\mathsf{C}, 0, f_\mathsf{T})$
V	$\sim$	$(f_\mathsf{A}, f_\mathsf{C}, f_\mathsf{G}, 0)$
N	$\sim$	$(f_\mathsf{A}, f_\mathsf{C}, f_\mathsf{G}, f_\mathsf{T})$

4.2 Model order

One essential input parameter of all tools is the model Order, which determines the maximal order of the iPMM, i.e., the maximal number of context positions to be taken into account for modeling intra-motif dependencies. The learning procedures of InMoDe attempt to avoid parameter overfitting by selecting sparse model structures, so even a high Order can be fruitful in principle, since higher-order dependencies are selected only if supported by the data and neglected otherwise. However, a large Order does have a significant effect on the running time of the structure learning algorithm, which is of particular relevance for the tools `DeNovoMoDe`, `MixtureMoDe`, and `FlexibleMoDe` that use a stochastic search and thus need to perform structure learning many times. Note that setting Order to 0 results in a traditional Position Weight Matrix (PWM) model [10].

4.3 Parameters for the stochastic search

All tools that perform a stochastic search (`DeNovoMoDe`, `MixtureMoDe`, and `FlexibleMoDe`) require three input parameters to control how exhaustive the search is supposed to be. They start the optimization algorithm R times, perform each time at least T iteration steps, and each time terminate only if there was no improvement of the score (Equation 5.3) during the last T' iteration steps. The parameter names in the tools thus correspond to the above-mentioned variables

- Initial iterations: T
- Additional iterations: T'
- Number of restarts: R

The default values ($T = 50, T' = 10, R = 3$) are relatively small, but in many cases they are already sufficient for a first glance on data in order to obtain initial results. Increasing Number of restarts is typically the most promising option to increase the probability of finding a hard-to-spot pattern.

4.4 Output names

All tools allow specifying a Name for saving the output. This Name is used as prefix of the names of individual results being produced, which is particularly useful within the `InMoDeGUI` interface in order to ensure that results of different analyses can be easily distinguished. If no Name is specified, a default name is constructed by several key input parameters. Detailed descriptions of the individual tools are available in Chapter 5.

5 Individual tools

This section contains a description of the individual tools that is widely identical with the tool-internal documentation.

5.1 SimpleMoDe

`SimpleMoDe` is a no-frills tool for learning model structure and parameters of a single iPMM of given maximal order from a set of pre-aligned DNA sequences of same length.

Parameters

Input data: The file with the input sequences (Section 4.1). All sequences are expected to be of same length.

Order: See Section 4.2. Default value is 2, but the tool terminates in a reasonable amount of time even with high orders such as 6 or 7, since structure learning is carried out only once.

Name: See Section 4.4. Default name is "iPMM(Order)", where Order stands for its chosen value.

Output

- an XML representation of the model that can be used in subsequent analyses with tools `ScanApp` or `ClassificationApp`.
- the parameter values in human-readable form.
- a sequence logo [7] of the position-specific mononucleotide marginals of the model.
- a sequence logo of the reverse complement of the position-specific mononucleotide marginals of the model.
- a conditional sequence logo (CSL) that visualizes all dependencies learned by the model in terms of context-specific conditional probabilities. It is a default variant and a more customized plot can be made with `VisualizationApp`. **Note:** a CSL is not returned if Order was set to 0.

To be precise...

Structure learning of the iPMM is done, for every position ℓ independently, by choosing a Parsimonious Context Tree (PCT) τ_ℓ that maximizes the BIC score [8]

$$S(\tau_\ell|\mathbf{x}) = \sum_{c\in\tau_\ell} 2N_{\ell ca}\log\left(\frac{N_{\ell ca}}{N_{\ell c\cdot}}\right) - |\tau_\ell|(|\mathcal{A}|-1)\log(N) \tag{5.1}$$

of the input data $\mathbf{x}$. The parameters are estimated according to the factorized sequential NML distribution [9]

$$\hat{\theta}_{\ell ca}(\mathbf{x}) = \frac{e(N_{\ell ca})(N_{\ell ca}+1)}{\sum_{b\in\mathcal{A}} e(N_{\ell cb})(N_{\ell cb}+1)}, \tag{5.2}$$

where $e(N) = (\frac{N+1}{N})^N$ for $N > 0$ and $e(0) = 1$.
This combination of structure score and parameter estimate has been shown to constitute a robust learning approach for iPMMs that does not require any (hyper)parameter tuning via e.g. cross validation. For further details see the corresponding publication [5].

5.2 DeNovoMoDe

`DeNovoMoDe` is a tool for performing a basic motif discovery in a set of long DNA sequences of possibly variable length based on a single iPMM as motif model. The promoter model follows the **o**ne **o**ccurrence **p**er **s**equence (OOPS) assumption with a single motif model and takes into account motif occurrences on both strands. If a search only on the forward strand is desired, if multiple motifs are to be learned, or if the input is weighted, use `FlexibleMoDe` instead. The learning algorithm is a stochastic search that uses the BIC score of the motif model as target function. For further details regarding the methodology see the corresponding publication [6].

Parameters

Input data: File with the input sequences (Section 4.1). They are allowed to differ in length.

Motif width: It may not exceed the length of the shortest input sequence.

Motif order: See Section 4.2. Default value is 2. While a higher value can be fruitful, the effects on runtime are here more severe compared to `SimpleMoDe`, and a significant increase in runtime typically appears when Order exceeds 3.

Flanking order: Order of the homogeneous Markov chain that models all parts of the sequence that are not covered by the motif. Default value is 2. The main purpose of the flanking model is to model low complexity repeats (mono-, di-, or trinucleotides) in order to avoid them to be erroneously identified as motif. As a consequence, a value larger than 3 is rarely justified.

Initial iterations, Additional iterations, Number of restarts: Parameters for controlling how exhaustive the stochastic search is supposed to be. For further details see Section 4.3.

Name: See Section 4.4. Default name is "DeNovo(Motif width,Motif order,Flanking order)".

Output

- A log-file containing the scores of all iteration steps in the stochastic search for evaluating whether the parameter values for Initial iterations, Additional iterations, and Number of restarts have been sufficient, or whether more effort could have been beneficial.
- The optimal latent variable configuration.
- The iPMM motif model with exactly the same output as returned by `SimpleMoDe`.

5.3 MixtureMoDe

`MixtureMoDe` is a tool for learning structure and parameters of a mixture of several iPMMs of from a set of pre-aligned and ungapped DNA sequences of the same length. It can be used to evaluate to which degree dependencies present in the data can also be represented as heterogeneities. The learning procedure is similar to that of `DeNovoMoDe`, but the target function is now the sum of the BIC scores of the individual components.

Parameters

Input data: The file with the input DNA sequences (Section 4.1), which are expected to have all the same length.

Number of mixture components: The number of iPMMs that are to be learned. Default value is 2. If the parameter is set to 1, the tool does essentially the same as `SimpleMoDe`.

Component order: See Section 4.2. It applies to every mixture component and the default value is 2. If different components are supposed to have a different maximal order, use `FlexibleMoDe` instead.

Initial iterations, Additional iterations, Number of restarts: The parameters for controlling how exhaustive the stochastic search is supposed to be. For further details see Section 4.3.

Name: See Section 4.4. Default name is "Mixture(a,b)", where a is Number of mixture components, and b is Component order.

Output

- A log-file containing the scores of all iteration steps in the stochastic search (See `DeNovoMoDe`).
- The optimal latent variable configuration.
- All learned components models, with each component containing exactly the same output as returned by `SimpleMoDe`.
  Note: The number of components returned can be smaller than Number of mixture components. In such a case, all components that are not returned did match no data points in the stochastic search. This can happen if Number of mixture components is chosen large in relation to sample size or if the data just does not justify the use of multiple components.

5.4 FlexibleMoDe

`FlexibleMoDe` contains the functionality of the previous three learning tools and allows various additional parameters to be set in order to learn iPMMs in more specialized scenarios. The underlying latent variable model contains – in its maximal variant – for every input sequence a latent variable for (a) motif position, (b) motif type, and (c) strand orientation and the optimization function for the stochastic search is the sum of all BIC scores over all motif types.

Parameters

Input data: The file with the input sequences (Section 4.1), which are allowed to differ in length.

Weights (optional): `FlexibleMoDe` supports learning on weighted data. The specified file must contain as many numbers (integer or double) as Input data has sequences. The numbers may be separated either by whitespace or by tabs and they are allowed to be spread over multiple lines. So files with (a) one line of N tab-separated numbers, or (b) N lines containing a single number, or (b) two lines of $\frac{N}{2}$ whitespace-separated numbers, etc. are all treated equally. If all given weights are equal to 1, the data is unweighted. This is also the default when no weight file is specified at all. Otherwise weight i is a multiplicative factor to the contribution of sequence i within the learning algorithm and the sum of all weights is interpreted as sample size.

Motif width: It may not exceed the length of the shortest input sequence.

Motif order(s): The parameter encodes both the number of motif types as well as their maximal order (Section 4.2). It expects a String of comma-separated integers. The number of integers determines the number of mixture components (of the motif model) and their values determine the corresponding model order. For example, if this parameters is set to *"1,2,0"*, a three-component mixture with different maximal model orders in each component is used for the motif.

Flanking order: It has the same meaning as in `DeNovoMoDe` and is thus only relevant iff at least one input sequence is longer than Motif width.

Both strands: Logical value that determines whether the putative binding sites can be present on both strands (true) or only on one strand (false).

Update motif type parameters: If true, the occurrence probability of every mixture component is dynamically updated during the iterative search. Otherwise every component is assumed to occur with equal probability.

Initial iterations, Additional iterations, Number of restarts: The parameters for controlling how exhaustive the stochastic search is supposed to be. For further details see Section 4.3.

Memoization: Logical value that determines whether the memoization technique [4] for speeding up structure learning should be used. It does not change the obtained result and can yield significant speedups when the model order is large. However, it can then also require substantially more RAM. Disabled by default.

Pruning: Logical value that determines whether pruning rules [3] for speeding up exact structure learning of PCTs should be used. They do not change the obtained result and typically yield significant speedups for highly structured data. However, in the worst case when data is near to uniform, such as in the early stages of an iterative search, pruning can slightly slow down the algorithm. Enabled by default.

Name: See Section 4.4. Default name "FlexibleMode".

Output

- A log-file containing the scores of all iteration steps in the stochastic search (see `DeNovoMoDe`).
- The optimal latent variable configuration.
- All learned motif/components models, with each component containing exactly the same output as returned by `SimpleMoDe`. **Note:** The number of returned components can be smaller than specified by Motif order(s) (See `MixtureMoDe`).

In order to obtain the exact functionality of the other three tools Weights are either not to be specified, or all of them have to equal 1, otherwise weighted-data versions of the learning algorithms are obtained. Apart from that, the following holds:

- If all sequences in Input data are of length Motif width, if Motif order(s) is set to a single number N, and Both strands is set to *false*, then the tool yields the same result as `SimpleMoDe` with Order set to N.
- If Motif order(s) is set to a single number N, and Both strands is set to *true*, then the tool does the same as `DeNovoMoDe` with Order set to N.
- If all sequences in Input data are of length Motif width, if Motif order(s) is set to a sequence of K equal, comma-separated integers N, and if Both strands is set to *false*, then the tool does the same as `MixtureMoDe` with Number of mixture components set to K and Component Order set to N.

All deviating parameter settings lead to scenarios that are not covered by the three less flexible tools. The parameters Memoization and Pruning affect only the time complexity of the structure learning and have no influence on the obtained result itself. By default, pruning is enabled and memoization is disabled in all tools.

To be precise...

Denote the input data by $\mathbf{x} = (\vec{x}_1, \ldots, \vec{x}_N)$, denote the latent variables of position, motif type, and strand orientation of a putative binding site within the i-th input sequence by u_i, v_i, s_i, denote the number of mixture components or the number of motif types by K, denote the Motif width by W, and denote the set of all model parameters by $\boldsymbol{\Theta}$. Then a single run of the algorithm in its unweighted variant is given by the following pseudo code:

for $t = 1, \ldots, T$ **do**
  for $i = 1, \ldots, N$ **do**
    sample $u_i^{(t)}$ from $\begin{cases} \left(\frac{1}{L_i - W + 1}, \ldots, \frac{1}{L_i - W + 1}\right) & \text{if } t = 1 \\ P(u_i | \vec{x}_i, \boldsymbol{\Theta}^{(t-1)}) & \text{if } t > 1 \end{cases}$
    sample $v_i^{(t)}$ from $\begin{cases} \left(\frac{1}{K}, \ldots, \frac{1}{K}\right) & \text{if } t = 1 \\ P(v_i | \vec{x}_i, \boldsymbol{\Theta}^{(t-1)}) & \text{if } t > 1 \end{cases}$
    sample $s_i^{(t)}$ from $\begin{cases} (0.5, 0.5) & \text{if } t = 1 \\ P(s_i | \vec{x}_i, \boldsymbol{\Theta}^{(t-1)}) & \text{if } t > 1 \end{cases}$
  end for
  for $k = 1, \ldots, K$ **do**
    obtain $\mathbf{y}_k^{(t)}$ as binding sites of component k in $\mathbf{x}$ according to latent variables $\vec{u}^{(t)}, \vec{v}^{(t)}, \vec{s}^{(t)}$
    for $\ell = 1, \ldots, W$ **do**
      compute $\tau_{k\ell}^{(t)} = \underset{\tau_\ell}{\operatorname{argmax}}\, S(\tau_{k\ell} | \mathbf{y}_k^{(t)})$ (Equation 5.1)
      for $c \in \tau_{k\ell}^{(t)}$ **do**
        estimate $\vec{\theta}_{k\ell c}^{(t)}$ from $\mathbf{y}_k^{(t)}$ (Equation 5.2)
      end for
    end for
  end for
  compute score

$$S^{(t)} = \sum_{k=1}^{K} S(\tau_{k\ell}^{(t)} | \mathbf{y}_k^{(t)}) \tag{5.3}$$

end for

If weights $\vec{w} = (w_1, \ldots, w_N)$ are provided, all computations normally involving counts of the data (Equation 5.1,5.2) now involve soft counts based on $\vec{w}$. The sample size N within the BIC score is replaced by $\sum_{i=1}^{N} w_i \delta_{v_i,k}$.

5.5 ScanApp

ScanApp scans a given data set for high-scoring occurrences of a previously learned iPMM. In order to assess which likelihood value is sufficient for declaring a hit, it determines a threshold based on a user-specified false positive rate (FPR) on a negative data set. A negative data set can be either given as user input or it is constructed as a randomized version of the positive data set.

Parameters

Input model: Any successfully learned iPMM (in XML representation) can be used and it does not matter which of the four learning tools has produced it.

Input data: The file with the DNA sequences (Section 4.1) in which the scan is to be performed.

Background: A selection parameter determining the choice of background data set to be used. Depending on the selection, additional parameters need to be specified:

- *Generating*: An artificial background data set will be constructed by learning a homogeneous Markov chain of user-specified Order from the Input data and then generating a sample from that chain. The size of the background data set equals the size of the input data set multiplied by the value of Size factor. Note that a larger background data set always yields a more accurate estimation of the desired threshold (in particular if the FPR is low), but the computation and sorting of all likelihood values can use a critical amount of time and memory.
- *From file*: The content of a user-specified Data file is used as background data set. The same format restrictions as for Input data apply (Section 4.1).

False positive rate: No matter whether the background is user-specified or generated, the tool then computes internally a likelihood-threshold for scoring putative binding sites the positive data that yields a given False positive rate in the background.

Both strands: If enabled (default), the scan is performed on both strands.

Name: See Section 4.4. Default name is "SequenceScan(False positive rate)".

Output

- Sequence ID, start position, end position, strand orientation, and likelihood-score of every hit in the Input data in a .BED-like format except that chromosome/scaffold name is here just the ID of the corresponding input sequence.
- The corresponding binding sites extracted from Input data, aligned, and put into the same strand orientation.

To be precise...

Let $\mathbf{x}$ denote Input data consisting of N sequences, let θ denote the parameters of Input model, let F denote Size factor, and let R denote the False positive rate. With Both strands enabled, the method works then as follows.

1. if Background is set to *Generating*:
   - learn parameters of a homogeneous Markov model θ_{bg} of user-specified order from $\mathbf{x}$
   - generate background dataset $\mathbf{y}$ from θ_{bg}, where $\mathbf{y}$ has the same number of sequences as $\mathbf{x}$ with the same length distribution
   - repeat the process $F-1$ times and add the generated sequences to $\mathbf{y}$ to finally obtain a background data set of size M

   if Background is set to *From file*: load $\mathbf{y}$ from Data file of size M
2. compute
   - $\forall_{i=1}^{M}\forall_{j=1}^{L_i-W+1} \log P(y_{i,j},\ldots,y_{i,j-W+1}|\theta)$ and
   - $\forall_{i=1}^{M}\forall_{j=1}^{L_i-W+1} \log P(\text{rc}(y_{i,j},\ldots,y_{i,j-W+1})|\theta)$ where rc is the function that returns the reverse complement of the sequence provided as argument,

   and compile all values in a list L
3. sort L in descending order
4. determine $T = L_{2RM}$ (i.e. T is the R-percentil of L)
5. scan $\mathbf{x}$:
   - $\forall_{i=1}^{N}\forall_{j=1}^{L_i-W+1} \log P(x_{i,j},\ldots,x_{i,j-W+1}|\theta) > T \rightarrow$ declare hit on forward strand
   - $\forall_{i=1}^{N}\forall_{j=1}^{L_i-W+1} \log P(\text{rc}(x_{i,j},\ldots,x_{i,j-W+1})|\theta) \rightarrow> T$ declare hit on reverse complementary strand

Note: The tool predicts solely based on the likelihood-scores, irrespective of putative overlaps among binding sites. If overlapping sites are not desired, the result has to be subsequently filtered by the user accordingly.

5.6 ClassificationApp

`ClassificationApp` performs a binary classification of sequences that can be scored by two iPMMs. Alternatively, it classifies sequences based on an iPMM as foreground model and a homogeneous Markov chain of given order that is learned on a given negative data set set.

Parameters

Input data: The file with the DNA sequences (Section 4.1) which are to be classified. All sequences in the data set need to be of same length W so that they can be scored by the given models.

Foreground model: Any learned iPMM (in XML representation) of width W can be used and it does not matter which of the learned tools has produced it.

Background: A selection parameter determining the choice of background model to be used. Depending on the selection, additional parameters might need to be specified:

- *Uniform PWM*: No additional parameters need to be set. The background model is a simple PWM model with all position-specific nucleotide probabilities equaling $\frac{1}{4}$.
- *Previously learned*: A background Model can be an iPMM (in XML representation) of possibly different order than Foreground model, but it must share the same width W.
- *Generating from data*: A homogeneous Markov chain of user-specified Order is learned from user-specified background Data. This model in then used as background model during the classification.

Name: See Section 4.4. Default name is "Classification".

Output

- The tool returns a list with pairs of sequence IDs (according to location in the input file) and class assignment.

5.7 VisualizationApp

`VisualizationApp` allows to plot a conditional sequence logo [2, 6] with customized layout from a previously learned iPMM. All parameters except for the input model are boolean (checkboxes) and determine whether particular plot elements are to be shown or not.

Parameters

Input model: Any successfully learned iPMM (in XML representation) can be used and it does not matter which of the four learning tools has produced it.

IUPAC: If enabled, all nodes in a PCT representing more than one symbol will be labeled by the corresponding IUPAC code.

Marginals: If disabled, the nucleotide stacks corresponding to the marginal distribution, which correspond to the traditional sequence logo, are hidden.

Trivial: If disabled, PCTs with only one leaf and corresponding nucleotide stacks are shown. While they do not contain any additional information compared to the traditional sequence logo, plotting them might be useful if Marginals is disabled.

Pseudonodes: If enabled, all pseudonodes (PCT nodes that are labeled by the full alphabet and have a trivial subtree) are plotted so that all PCT leaves are located at the maximal depth.

IC scaled: If disabled, the nucleotide stacks are not scaled according to their information content, which can be helpful to perceive small differences in distributions that are rather close to uniform.

Y labels: If enabled, short descriptions of the different plot elements are shown on the left near the Y axes.

Grid: If disabled, the grid separating the individual plot elements is not shown.

Positions: If disabled, the sequence position labels at the top of the plot is not shown.

Context labels: If disabled, the individual context numbers are not plotted.

Output

- The tool returns a plot of a CSL with the desired properties.

Bibliography

[1] R. Eggeling, A. Gohr, P.-Y. Bourguignon, E. Wingender, and I. Grosse. Inhomogeneous parsimonious Markov models. In *Proceedings of ECMLPKDD*, volume 1, pages 321–336. Springer, 2013.

[2] R. Eggeling, A. Gohr, J. Keilwagen, M. Mohr, S. Posch, A.D. Smith, and I. Grosse. On the value of intra-motif dependencies of human insulator protein CTCF. *PLOS ONE*, 9(1):e85629, 2014.

[3] R. Eggeling and M. Koivisto. Pruning rules for learning parsimonious context trees. In *Proceedings of UAI*, volume 32, pages 152–161. AUAI press, 2016.

[4] R. Eggeling, M. Koivisto, and I. Grosse. Dealing with small data: On the generalization of context trees. In *Proceedings of ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, 2015.

[5] R. Eggeling, T. Roos, P. Myllymäki, and I. Grosse. Robust learning of inhomogeneous PMMs. In *Proceedings of AISTATS*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 229–237, 2014.

[6] R. Eggeling, T. Roos, P. Myllymäki, and I. Grosse. Inferring intra-motif dependencies of DNA binding sites from ChIP-seq data. *BMC Bioinform.*, 16:375, 2015.

[7] T.D. Schneider and R.M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res.*, 18(20):6097–6100, 1990.

[8] Gideon E. Schwarz. Estimating the dimension of a model. *Ann. Stat.*, 2:461–464, 1978.

[9] T. Silander, T. Roos, and P. Myllymäki. Locally minimax optimal predictive modeling with Bayesian networks. In *Proceedings of AISTATS*, volume 5 of *JMLR Workshop and Conference Proceedings*, pages 504–511, 2009.

[10] G.D. Stormo, T.D Schneider, and L.M. Gold. Characterization of translational initiation sites in E.coli. *Nucleic Acids Res.*, 10(2):2971–2996, 1982.