SECOMO Documentation

Release 1.1.0

Roman Schulte-Sasse, Wolfgang Kopp

Oct 02, 2017

Contents:

CHAPTER 1

Introduction to SECOMO: A SEquence COntext MOdeler

This package provides functionality to automatically extract DNA sequence features from a provided set of sequences using a **convolutional restricted Boltzmann machine**, which was inspired by advances made in computer vision[1]. To that end, the cRBM learns redundant DNA sequnce features in terms of a set of weight matrices.

For a downstream analysis of the features that have been learned a number of utilities are provided:

1. Convert the cRBM features to **Position frequency matrices**, which are commonly used to represent transcription factor binding affinities[2].
2. Visualize the DNA features in terms of **Sequence logos**[2].
3. Visualize the **positional enrichment** of the features on a set of DNA sequences.
4. Visualize the **relative enrichment** of the features across a number of different datasets (e.g. sequences of different ChIP-seq experiments; treatment-control).
5. Visualize sequence-based **clustering**.

The tutorial illustrates the main functionality of the package on a toy example of *Oct4* ChIP-seq peak regions obtained from embryonic stem cells.

Finally, if this tool helps for your analysis, please cite the package:

```
@Manual{,
    title = {SECOMO: A python package for automatically
             extracting DNA sequence features},
    author = {Roman Schulte-Sasse, Wolfgang Kopp},
    year = {2017},
}
```

References

[1] Lee, Honglak (2009). Convolutional Deep Belief Networks for scalable unsupervised learning of hierarchical representations. Proceedings of the 26 th International Confefence on Machine Learning

[2] Stormo, Gary D. (2000). DNA binding sites: representation and discovery. Bioinformatics.

CHAPTER 2

System setup

Requirements

The `secomo` package is compatible and was tested with py2.7, py3.4, py3.5 and py3.5. Prerequisites:

- `theano`
- `numpy`
- `scipy`
- `joblib`
- `Biopython`
- `matplotlib`
- `pandas`
- `weblogolib`
- `scikit-learn`
- `seaborn`

The latter collection of packages are necessary to investigate the model results using our set of provided functions (e.g. `secomo.utils.scatterTSNE()`).

Finally, if possible, we recommend utilizing CUDA. *Theano* take advantage of *cuda*, which significantly speeds up the training phase. See Theano documention for more information.

Installation

The secomo package can be obtained from GitHub

At the moment you can install it according to:

```
git clone https://github.com/schulter/crbm.git
cd crbm/
python setup.py install
```

CHAPTER 3

Tutorial

This section is intended to demonstrate the main functionality of SECOMO on a small toy dataset.

Loading sample dataset

Assuming you have successfully install the *secomo* package, you can load a sample dataset consisting of *Oct4* ChIP-seq peaks from embryonic stem cells (ESCs)

```
import secomo

# Obtain sample sequences in one-hot encoding
onehot = secomo.load_sample()
```

The original DNA sequences first need to be converted to *one-hot* encoding

Note: The sample sequences are contained in the *secomo* package in fasta format. Generally, fasta files can be loaded and converted to *one-hot* encoding according to

```
# Convert to one-hot
seqs = secomo.readSeqsFromFasta("path/to/seq.fa")
onehot = secomo.seqToOneHot(seqs)
```

Train SECOMO's convolutional RBM model

Next, we instantiate a cRBM to learn 10 motifs of length 15 bp and train it on the provided sequences

```
# Obtain a cRBM object
model = secomo.CRBM(num_motifs = 10, motif_length = 15)
```

```
# Fit the model
model.fit(onehot)
```

Note: The *CRBM* object can be instantiated with a number of training-related hyper-parameters, e.g. number of epochs. See API for more information.

Note: Optionally, the *fit* method also accepts a validation dataset on which the training progress is monitored. This makes it easier to detect overfitting. If no validation set is supplied, the progress is reported on the training set.

Save and restore the parameters

After having trained the model, it is common to store the parameters reused them later for a subsequent analysis. To this end, the following methods can be invoked

```
# Save parameters and hyper-parameters
model.saveModel('oct4_model_params.pkl')

# Reinstantiate model
model = secomo.CRBM.loadModel('oct4_model_params.pkl')
```

Position frequency matrices

A common way to investigate patterns in DNA sequences is to model them as *position frequency matrices* (PFMs) which can be then nicely visualized. The model parameters (e.g. the weight matrices) learned by the cRBM can be converted to such PFMs, which can then be used for further downstream analysis. For this purpose one can utilize

```
# Get a list of numpy matrices representing PFMs
model.getPFMs()

# Store the PFMs (by default in 'jaspar' format)
# in the folder './pfms/'
secomo.saveMotifs(model, path = './pfms/')
```

PFMs are frequently visualized in terms of sequence logos which can be obtained by

```
# Writes all logos in the logos/ directory
secomo.utils.createSeqLogos(model, path = "./logos/")

# Alternatively, an individual sequence logo can be created:
# Get first motif
pfm = model.getPFMs()[0]

# Create a corresponding sequence logo
secomo.utils.createSeqLogo(pfm, filename = "logo1.png", fformat = "png")
```

Motif matches

Next, we inspect at which positions in a set of DNA sequences motif matches are present. The per-position motif match probabilities can be obtained as follows

```
# Per-position motif match probabilities
# for the first 100 sequences
matches = model.motifHitProbs(onehot[:100])
```

Here, `matches` represents a 4D numpy array comprising the match probabilities with dimensions *Nseqs x num_motifs x 1 x (Seqlengths - motif_length + 1)*.

An average profile of match probabilities per-position can be illustrated using

```
# Plot positional enrichment for all motifs in the given
# test sequences
secomo.positionalDensityPlot(model, onehot[:100], filename = './densityplot.png')
```

Clustering analysis

Finally, we shall demonstrate that the sequence activations of similar sequences tend to cluster together. In order to visualize the clusters, we run TSNE dimensionality reduction using

```
# Run t-SNE dim reduction to 2D
tsne = secomo.runTSNE(model, onehot)

# Visualize the results in a scatter plot
secomo.tsneScatter({'Oct4': tsne}, filename = './tsnescatter.png')

# Visualize the results in the scatter plot
# by augmenting with the respective motif abundances
secomo.tsneScatterWithPies(model, onehot, tsne, filename = "./tsnescatter_pies.png")
```

Motif enrichment across different sets of sequences

This part concerns the analysis of multiple datasets with the same cRBM. In order to find out whether a specific motif (e.g. weight matrices) is enriched or depleted in a certain dataset relative to the others a violin plot can be created. In the following example, we just artificially split the *Oct4* dataset into *set1* and *set2* to illustrate the function

```
# Assemble multiple datasets as follows
data = {'set1': onehot[:1500], 'set2': onehot[1500:]}

secomo.violinPlotMotifMatches(model, data,
        filename = os.path.join(path, 'violinplot.png'))
```

Summary of the full analysis

The full tutorial code can be found in the Github repository: crbm/tutorial.py

=== SECOMO API ===

CHAPTER 4

Convolutional restricted Boltzmann machine (cRBM)

CRBM contains the main functionality of SECOMO for training, evaluating and investigating models.

CRBM.fit(training_data[, test_data])	Fits the cRBM to the provided training sequences.
CRBM.freeEnergy(data)	Free energy determined on the given dataset.
CRBM.motifHitProbs(data)	Motif match probabilities.
CRBM.getPFMs()	Returns the weight matrices converted to *position frequency matrices*.
CRBM.saveModel(filename)	Save the model parameters and additional hyperparameters.
CRBM.loadModel(filename)	Load a model from a given pickle file.

class secomo.**CRBM**(*num_motifs*, *motif_length*, *epochs=100*, *input_dims=4*, *doublestranded=True*, *batchsize=20*, *learning_rate=0.1*, *momentum=0.95*, *pooling=1*, *cd_k=5*, *rho=0.01*, *lambda_rate=0.1*)

CRBM class.

The class *CRBM* implements functionality for a *convolutional restricted Boltzmann machine* (cRBM) that extracts redundant DNA sequence features from a provided set of sequences. The model can subsequently be used to study the sequence content of (e.g. regulatory) sequences, by visualizing the features in terms of sequence logos or in order to cluster the sequences based on sequence content.

num_motifs [int] Number of motifs.

motif_length [int] Motif length.

epochs [int] Number of epochs to train (Default: 100).

input_dims :int Input dimensions aka alphabet size (Default: 4 for DNA).

doublestranded [bool] Single strand or both strands. If set to True, both strands are scanned. (Default: True).

batchsize [int] Batch size (Default: 20).

learning_rate [float)] Learning rate (Default: 0.1).

momentum [float] Momentum term (Default: 0.95).

pooling [int] Pooling factor (not relevant for cRBM, but for future work) (Default: 1).

cd_k [int] Number of Gibbs sampling iterations in each persistent contrastive divergence step (Default: 5).

rho [float] Target frequency of motif occurrences (Default: 0.01).

lambda_rate [float] Sparsity enforcement aka penality term (Default: 0.1).

fit(*training_data*, *test_data=None*)
Fits the cRBM to the provided training sequences.

training_data [numpy-array] 4D-Numpy array representing the training sequence in one-hot encoding. See `crbm.sequences.seqToOneHot()`.

test_data [numpy-array] 4D-Numpy array representing the validation sequence in one-hot encoding. If no test_data is provided, the training progress will be reported on the training set itself. See `crbm.sequences.seqToOneHot()`.

freeEnergy(*data*)
Free energy determined on the given dataset.

data [numpy-array] 4D numpy array representing a DNA sequence in one-hot encoding. See `crbm.sequences.seqToOneHot()`.

returns [numpy-array] Free energy per sequence.

getPFMs()
Returns the weight matrices converted to *position frequency matrices*.

returns: numpy-array List of position frequency matrices as numpy arrays.

classmethod loadModel(*filename*)
Load a model from a given pickle file.

filename [str] Pickle file containing the model parameters.

returns [*CRBM* object] An instance of CRBM with reloaded parameters.

motifHitProbs(*data*)
Motif match probabilities.

data [numpy-array] 4D numpy array representing a DNA sequence in one-hot encoding. See `crbm.sequences.seqToOneHot()`.

returns [numpy-array] Per-position motif match probabilities of all motifs as numpy array.

saveModel(*filename*)
Save the model parameters and additional hyper-parameters.

filename [str] Pickle filename where the model parameters are stored.

CHAPTER 5

Sequence-related utilies

Functions for loading DNA sequences in fasta format and for convert them to the required *one-hot* encoding.

readSeqsFromFasta(filename)	Read sequences from multi-fasta file.
seqToOneHot(seqs)	Converts a set of Biopython DNA sequences to *one-hot* encoding.
splitTrainingTest(filename, train_test_ratio)	Splits training and test set.

secomo.sequences.**readSeqsFromFasta**(*filename*)

Read sequences from multi-fasta file.

filename [str] File containing DNA sequences in multi-fasta format.

returns [list] List of Biopython Seq objects.

secomo.sequences.**seqToOneHot**(*seqs*)

Converts a set of Biopython DNA sequences to *one-hot* encoding.

seqs :list List of Biopython Sequence objects.

returns [numpy-array] 4D Numpy array representing the sequences in *one-hot* encoding.

secomo.sequences.**splitTrainingTest**(*filename*, *train_test_ratio*, *num_top_regions=None*, *randomize=True*)

Splits training and test set.

The DNA sequences are read from filename. As output, two additional fasta files are generated with prefixes being same as filename and suffixes *train.fa* and *test.fa*.

filename [str] File containing DNA sequences in multi-fasta format.

train_test_ratio [float] Ratio between 0 and 1 for splitting the dataset into training and test. For instance, train_test_ratio=0.1 uses 10% and 90% of the dataset for test and training, respectively.

num_top_regions [int] Use only the first num_top_regions from the fasta-file. Default: (None) all sequences are used.

randomize [bool] Randomize the dataset. Default: True.

CHAPTER 6

Utils

This part presents functions contained in *secomo.utils* that help you investigate the results of a trained SECOMO model. It features generating position frequency matrices, sequence logos and clustering plots.

saveMotifs(model, path[, name, fformat])	Save weight-matrices as PFMs.
createSeqLogos(model, path[, fformat])	Create sequence logos for all cRBM motifs
createSeqLogo(pfm, filename[, fformat])	Create sequence logo for an individual cRBM motif.
positionalDensityPlot(model, seqs[, filename])	Positional enrichment of the motifs.
runTSNE(model, seqs)	Run t-SNE on the motif abundances.
tsneScatter(data[, lims, colors, filename, ...])	Scatter plot of t-SNE clustering.
tsneScatterWithPies(model, seqs, tsne[, ...])	Scatter plot of t-SNE clustering.
violinPlotMotifMatches(model, data[, filename])	Violin plot of motif abundances.

secomo.utils.**saveMotifs**(*model*, *path*, *name='mot'*, *fformat='jaspar'*)

Save weight-matrices as PFMs.

This method converts the cRBM weight-matrices to position frequency matrices and stores each matrix in a single file with ending .pfm.

model [CRBM object] A cRBM object.

path [str] Directory in which the PFMs should be stored.

name [str] File prefix for the motif files. Default: 'mot'.

fformat [str] File format of the motifs. Either 'jaspar' or 'tab'. Default: 'jaspar'.

secomo.utils.**positionalDensityPlot**(*model*, *seqs*, *filename=None*)

Positional enrichment of the motifs.

This function creates a figure that illustrates a positional enrichment of all cRBM motifs in the given set of sequences.

model [CRBM object] A cRBM object

seqs [numpy-array] A set of DNA sequences in *one-hot* encoding. See crbm.sequences.seqToOneHot()

filename [str] Filename for storing the figure. If `filename = None`, no figure will be stored.

`secomo.utils.`**`runTSNE`**(*model*, *seqs*)
Run t-SNE on the motif abundances.

This function produces a clustering of the sequences using t-SNE based on the motif matches in the sequences. Accordingly, the sequences are projected onto a 2D hyper-plane in which similar sequences are located in close proximity.

model [`CRBM` object] A cRBM object

seqs [numpy-array] A set of DNA sequences in *one-hot* encoding. See `crbm.sequences.seqToOneHot()`

`secomo.utils.`**`tsneScatter`**(*data*, *lims=None*, *colors=None*, *filename=None*, *legend=True*)
Scatter plot of t-SNE clustering.

data [dict] Dictionary containing the dataset name (keys) and data itself (values). The data is assumed to have been generated using *`runTSNE()`*.

lims [tuple] Optional parameter containing the x- and y-limits for the figure. If None, the limits are automatically determined. For example: `lims = ([xmin, ymin], [xmax, ymax])`

colors [matplotlib.cm] Optional colormap to illustrate the datapoints. If None, a default colormap will be used.

filename [str] Filename for storing the figure. Default: None, means the figure stored but directly displayed.

legend [bool] Include the legend into the figure. Default: True

`secomo.utils.`**`createSeqLogos`**(*model*, *path*, *fformat='eps'*)
Create sequence logos for all cRBM motifs

model [`CRBM` object] A cRBM object.

path [str] Output folder.

fformat [str] File format for storing the sequence logos. Default: 'eps'.

`secomo.utils.`**`createSeqLogo`**(*pfm*, *filename*, *fformat='eps'*)
Create sequence logo for an individual cRBM motif.

pfm [numpy-array] 2D numpy array representing a PFM. See `CRBM.getPFMs()`

path [str] Output folder.

fformat [str] File format for storing the sequence logos. Default: 'eps'.

`secomo.utils.`**`tsneScatterWithPies`**(*model*, *seqs*, *tsne*, *lims=None*, *filename=None*)
Scatter plot of t-SNE clustering.

This function produces a figure in which sequences are represented as pie chart in a 2D t-SNE hyper-plane obtained with *`runTSNE()`*. Moreover, the pie pieces correspond to the individual `CRBM` motifs and the sizes represent the enrichment/abundance of the motifs in the respective sequences.

model [`CRBM` object] A cRBM object

seqs [numpy-array] DNA sequences represented in *one-hot* encoding. See `crbm.sequences.seqToOneHot()`.

tsne [numpy-array] 2D numpy array representing the sequences projected onto the t-SNE hyper-plane that was obtained with *`runTSNE()`*.

lims [tuple] Optional parameter containing the x- and y-limits for the figure. For example:

([xmin, ymin], [xmax, ymax])

filename [str] Filename for storing the figure.

secomo.utils.**violinPlotMotifMatches**(*model*, *data*, *filename=None*)

Violin plot of motif abundances.

This function summarized the relative motif abundances of the CRBM motifs in a given set of sequences (e.g. sequences with different functions).

model [CRBM object] A cRBM object

data [dict] Dictionary with keys representing dataset-names and values a set of DNA sequences in *one-hot* encoding. See crbm.sequences.seqToOneHot().

filename [str] Filename to store the figure. Default: None, the figure will be dirctly displayed.

CHAPTER 7

Sample dataset

The package contains a small sample dataset consisting of *Oct4* ChIP-seq sequences of embryonic stem cells from ENCODE[1].

secomo.sequences.**load_sample**()

Load sample sequences of Oct4 ChIP-seq peaks from H1hesc cell of ENCODE. The sequences are converted to one-hot encoding.

returns [numpy-array] Sample DNA sequences in *one-hot* encoding.

[1] ENCODE Project Consortium and others. (2012). An integrated encyclopedia of DNA elements in the human genome. Nature.

CHAPTER 8

Acknowledgements

This project was developed by **Roman Schulte-Sasse** (@schulter) as part of his Master thesis with help and under the supervision of **Wolfgang Kopp** (@wkopp) and with co-supervision of **Prof. Dr. Annalisa Marsico** at the *Max Planck Institute for molecular genetics*. The code was later adapted for the publication by Wolfgang with help of Roman and is currently maintained by Roman and Wolfgang.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index