SuSi Documentation

Release 1.1.0

Felix M. Riese

Aug 31, 2020

CONTENTS

We present the SuSi package for Python. It includes a fully functional SOM for unsupervised, supervised and semi-supervised tasks.

License 3-Clause BSD license

Author Felix M. Riese

Citation see Citation and in the bibtex file

Paper F. M. Riese, S. Keller and S. Hinz in Remote Sensing, 2020

CHAPTER

ONE

CHANGE LOG

1.1 [1.1.0] - 2020-08-31

- [ADDED] Logo.
- [ADDED] SOMPlots documentation.
- [REMOVED] Python 3.5 support. Now, only 3.6-3.8 are supported.
- [FIXED] Scikit-learn warnings regarding validation of positional arguments.
- [FIXED] Sphinx documentation warnings.

1.2 [1.0.10] - 2020-04-21

- [ADDED] Support for Python 3.8.x.
- [ADDED] Test coverage and MultiOutput test.
- [CHANGED] Function *setPlaceholder* to *set_placeholder*.
- [FIXED] Documentation links

1.3 [1.0.9] - 2020-04-07

- [ADDED] Documentation of the hyperparameters.
- [ADDED] Plot scripts.
- [CHANGED] Structure of the module files.

1.4 [1.0.8] - 2020-01-20

- [FIXED] Replaced scikit-learn *sklearn.utils.fixes.parallel_helper*, see #12.

1.5 [1.0.7] - 2019-11-28

- [ADDED] Optional tqdm visualization of the SOM training
- [ADDED] New *init_mode_supervised* called *random_minmax*.
- [CHANGED] Official name of package changes from *SUSI* to *SuSi*.
- [CHANGED] Docstrings for functions are now according to guidelines.
- [FIXED] Semi-supervised classification handling, sample weights
- [FIXED] Supervised classification SOM initalization of *n_iter_supervised*
- [FIXED] Code refactored according to prospector
- [FIXED] Resolved bug in *get_datapoints_from_node()* for unsupervised SOM.

1.6 [1.0.6] - 2019-09-11

- [ADDED] Semi-supervised abilities for classifier and regressor
- [ADDED] Example notebooks for semi-supervised applications
- [ADDED] Tests for example notebooks
- [CHANGED] Requirements for the SuSi package
- [REMOVED] Support for Python 3.4
- [FIXED] Code looks better in documentation with sphinx.ext.napoleon

1.7 [1.0.5] - 2019-04-23

- [ADDED] PCA initialization of the SOM weights with 2 principal components
- [ADDED] Variable variance
- [CHANGED] Moved installation guidelines and examples to documentation

1.8 [1.0.4] - 2019-04-21

- [ADDED] Batch algorithm for unsupervised and supervised SOM
- [ADDED] Calculation of the unified distance matrix (u-matrix)
- [FIXED] Added estimator_check of scikit-learn and fixed recognized issues

1.9 [1.0.3] - 2019-04-09

- [ADDED] Link to arXiv paper
- [ADDED] Mexican-hat neighborhood distance weight
- [ADDED] Possibility for different initialization modes
- [CHANGED] Simplified initialization of estimators
- [FIXED] URLs and styles in documentation
- [FIXED] Colormap in Salinas example

1.10 [1.0.2] - 2019-03-27

- [ADDED] Codecov, Codacy
- [CHANGED] Moved decreasing_rate() out of SOM classes
- [FIXED] Removed duplicate constructor for SOMRegressor, fixed fit() params

1.11 [1.0.1] - 2019-03-26

- [ADDED] Config file for Travis
- [ADDED] Requirements for read-the-docs documentation

1.12 [1.0.0] - 2019-03-26

- Initial release

CHAPTER

TWO

CITATION

The bibtex file including both references is available here.

2.1 Paper citation

F. M. Riese, S. Keller and S. Hinz, "Supervised and Semi-Supervised Self-Organizing Maps for Regression and Classification Focusing on Hyperspectral Data", *Remote Sensing*, vol. 12, no. 1, 2020.

```
@article{riese2020supervised,
    author = {Riese, Felix~M. and Keller, Sina and Hinz, Stefan},
    title = {{Supervised and Semi-Supervised Self-Organizing Maps for
              Regression and Classification Focusing on Hyperspectral Data}},
    journal = {Remote Sensing},
    year = {2020},
    volume = {12},
    number = {1},
    article-number = {7},
    URL = {https://www.mdpi.com/2072-4292/12/1/7},
    ISSN = {2072-4292},
    DOI = {10.3390/rs12010007}
}
```

2.2 Code citation

Felix M. Riese, "SuSi: SUpervised Self-organIzing maps in Python", 10.5281/zenodo.2609130, 2019.

```
@misc{riese2019susicode,
    author = {Riese, Felix~M.},
    title = {{SuSi: SUpervised Self-organIzing maps in Python}},
    year = {2019},
    DOI = {10.5281/zenodo.2609130},
    publisher = {Zenodo},
    howpublished = {\href{https://doi.org/10.5281/zenodo.2609130}{doi.org/10.5281/
↪zenodo.2609130}}
}
```

CHAPTER

THREE

INSTALLATION

3.1 Dependencies

Install Python 3, e.g. via Anaconda.

Install the required packages:

conda install –file requirements.txt

3.2 Install via PyPi

```
pip3 install susi
```

3.3 Install without PyPi

1. Download the package on PyPi
2. Navigate to the install folder in your terminal with `cd`
3. Install with `python setup.py install`

3.4 Install latest development version

```
git clone https://github.com/felixriese/susi.git
cd susi/
python setup.py install
```

CHAPTER

FOUR

EXAMPLES

4.1 Regression example

In `python3`:

```
import susi

som = susi.SOMRegressor()
som.fit(X_train, y_train)
print(som.score(X_test, y_test))
```

4.2 Classification example

In `python3`:

```
import susi

som = susi.SOMClassifier()
som.fit(X_train, y_train)
print(som.score(X_test, y_test))
```

4.3 More examples

- examples/SOMClustering
- examples/SOMRegressor
- examples/SOMRegressor_semisupervised
- examples/SOMClassifier
- examples/SOMClassifier_semisupervised

CHAPTER

FIVE

HYPERPARAMETERS

In the following, the most important hyperparameters of the SuSi package are described. The default hyperparameter settings are a good start, but can always be optimized. You can do that yourself or through an optimization. The commonly used hyperparameter settings are taken from [RieseEtAl2020].

5.1 Grid Size (n_rows, n_columns)

The grid size of a SOM is defined with the parameters `n_rows` and `n_columns`, the numbers of rows and columns. The choice of the grid size depends on several trade-offs.

Characteristics of a larger grid:

- Better adaption on complex problems (good!)
- Better/smoother visualization capabilities (good!)
- More possible overtraining (possibly bad)
- Larger training time (bad if very limited resources)

Our Recommendation: We suggest to start with a small grid, meaning `5 x 5`, and extending this grid size while tracking the test and training error metrics. We consider SOM grids as "large" with a grid size of about `100 x 100` and more. Non-square SOM grids can also be helpful for specific problems. Commonly used grid sizes are `50 x 50` to `80 x 80`.

5.2 Number of iterations and training mode

The number of iterations (`n_iter_unsupervised` and `n_iter_supervised`) depends on the training mode (`train_mode_unsupervised` and `train_mode_supervised`).

Our Recommendation (Online Mode) Use the *online* mode. If your dataset is small (< 1000 datapoints), use 10 000 iterations for the unsupervised SOM and 5000 iterations for the supervised SOM as start values. If your dataset is significantly larger, use significantly more iterations. Commonly used value ranges are for the unsupervised SOM 10 000 to 60 000 and for the (semi-)supervised SOM about 20 000 to 70 000 in the *online* mode.

Todo: Add recommendations for the batch mode.

5.3 Neighborhood Distance Weight, Neighborhood Function, and Learning Rate

The hyperparameters around the neighborhood mode (`neighborhood_mode_unsupervised` + `neighborhood_mode_supervised`) and the learning rate (`learn_mode_unsupervised`, `learn_mode_supervised`, `learning_rate_start`, and `learning_rate_end`) depend on the neighborhood distance weight formula `nbh_dist_weight_mode`. Two different modes are implemented so far: `pseudo-gaussian` and `mexican-hat`.

Our Recommendation (Pseudo-Gaussian): Use the `pseudo-gaussian` neighborhood distance weight with the default formulas for the neighborhood mode and the learning rate. The most influence, from our experiences, comes from the start (and end) value of the learning rate (`learning_rate_start`, and `learning_rate_end`). They should be optimized. Commonly used formula are `linear` and `min` for the neighborhood mode, `min` and `exp` for the learning rate mode, start values from 0.3 to 0.8 and end values from 0.1 to 0.005.

Todo: Add recommendations for the mexican hat distance weight.

5.4 Distance Metric

In the following, we give recommendations for the distance metric.

Todo: Add recommendations for the distance metric.

5.5 Hyperparameter optimization

Possible ways to find optimal hyperparameters for a problem are a grid search or randomized search. Because the SuSi package is developed according to several scikit-learn guidelines, it can be used with:

- scikit-learn.model_selection.GridSearchCV
- scikit-learn.model_selection.RandomizedSearchCV

For example, the randomized search can be applied as follows in `Python3`:

```
import susi
from sklearn.datasets import load_iris
from sklearn.model_selection import RandomizedSearchCV

iris = load_iris()
param_grid = {
    "n_rows": [5, 10, 20],
    "n_columns": [5, 20, 40],
    "learning_rate_start": [0.5, 0.7, 0.9],
    "learning_rate_end": [0.1, 0.05, 0.005],
}
som = susi.SOMRegressor()
clf = RandomizedSearchCV(som, param_grid, random_state=1)
```

(continues on next page)

(continued from previous page)

```
clf.fit(iris.data, iris.target)
print(clf.best_params_)
```

5.6 References

CHAPTER

SIX

SOMCLUSTERING

class susi.**SOMClustering**(*n_rows: int = 10*, *n_columns: int = 10*, *init_mode_unsupervised: str = 'random'*, *n_iter_unsupervised: int = 1000*, *train_mode_unsupervised: str = 'online'*, *neighborhood_mode_unsupervised: str = 'linear'*, *learn_mode_unsupervised: str = 'min'*, *distance_metric: str = 'euclidean'*, *learning_rate_start=0.5*, *learning_rate_end=0.05*, *nbh_dist_weight_mode: str = 'pseudo-gaussian'*, *n_jobs=None*, *random_state=None*, *verbose=0*)

Unsupervised self-organizing map for clustering.

Parameters

- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **init_mode_unsupervised** (*str, optional (default="random")*) – Initialization mode of the unsupervised SOM
- **n_iter_unsupervised** (*int, optional (default=1000)*) – Number of iterations for the unsupervised SOM
- **train_mode_unsupervised** (*str, optional (default="online")*) – Training mode of the unsupervised SOM
- **neighborhood_mode_unsupervised** (*str, optional (default="linear")*) – Neighborhood mode of the unsupervised SOM
- **learn_mode_unsupervised** (*str, optional (default="min")*) – Learning mode of the unsupervised SOM
- **distance_metric** (*str, optional (default="euclidean")*) – Distance metric to compare on feature level (not SOM grid). Possible metrics: {"euclidean", "manhattan", "mahalanobis", "tanimoto"}. Note that "tanimoto" tends to be slow.
- **learning_rate_start** (*float, optional (default=0.5)*) – Learning rate start value
- **learning_rate_end** (*float, optional (default=0.05)*) – Learning rate end value (only needed for some lr definitions)
- **nbh_dist_weight_mode** (*str, optional (default="pseudo-gaussian")*) – Formula of the neighborhood distance weight. Possible formulas are: {"pseudo-gaussian", "mexican-hat"}.
- **n_jobs** (*int or None, optional (default=None)*) – The number of jobs to run in parallel.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

- **verbose** (*int, optional (default=0)*) – Controls the verbosity.

Variables

- **node_list_** (*np.array of (int, int) tuples*) – List of 2-dimensional coordinates of SOM nodes
- **radius_max_** (*float, int*) – Maximum radius of the neighborhood function
- **radius_min_** (*float, int*) – Minimum radius of the neighborhood function
- **unsuper_som_** (*np.array*) – Weight vectors of the unsupervised SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **X_** (*np.array*) – Input data
- **fitted_** (*boolean*) – States if estimator is fitted to X
- **max_iterations_** (*int*) – Maximum number of iterations for the current training
- **bmus_** (*list of (int, int) tuples*) – List of best matching units (BMUs) of the dataset X
- **variances_** (*array of float*) – Standard deviations of every feature

calc_learning_rate(*curr_it*, *mode*)

Calculate learning rate alpha with 0 <= alpha <= 1.

Parameters

- **curr_it** (*int*) – Current iteration count
- **mode** (*str, optional*) – Mode of the learning rate (min, exp, expsquare)

Returns Learning rate

Return type float

calc_neighborhood_func(*curr_it*, *mode*)

Calculate neighborhood function (= radius).

Parameters

- **curr_it** (*int*) – Current number of iterations
- **mode** (*str*) – Mode of the decreasing rate

Returns Neighborhood function (= radius)

Return type float

calc_u_matrix_distances()

Calculate the Eucl. distances between all neighbored SOM nodes.

calc_u_matrix_means()

Calculate the missing parts of the u-matrix.

After *calc_u_matrix_distances()*, there are two kinds of entries missing: the entries at the positions of the actual SOM nodes and the entries in between the distance nodes. Both types of entries are calculated in this function.

fit(*X*, *y=None*)

Fit unsupervised SOM to input data.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The training input samples.

- **y** (*None*) – Not used in this class.

Returns self

Return type object

Examples

Load the SOM and fit it to your input data *X* with:

```
>>> import susi
>>> som = susi.SOMClustering()
>>> som.fit(X)
```

fit_transform(*X*, *y=None*)
Fit to the input data and transform it.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The training and prediction input samples.
- **y** (*None, optional*) – Ignored.

Returns Predictions including the BMUs of each datapoint

Return type np.array of tuples (int, int)

Examples

Load the SOM, fit it to your input data *X* and transform your input data with:

```
>>> import susi
>>> som = susi.SOMClustering()
>>> X_transformed = som.fit_transform(X)
```

get_bmu(*datapoint*, *som_array*)
Get best matching unit (BMU) for datapoint.

Parameters

- **datapoint** (*np.array, shape=shape[1]*) – Datapoint = one row of the dataset X
- **som_array** (*np.array*) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])

Returns Position of best matching unit (row, column)

Return type tuple, shape = (int, int)

get_bmus(*X*, *som_array=None*)
Get Best Matching Units for big datalist.

Parameters

- **X** (*np.array*) – List of datapoints
- **som_array** (np.array, optional (default=`None`)) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])

Returns **bmus** – Position of best matching units (row, column) for each datapoint

Return type list of (int, int) tuples

Examples

Load the SOM, fit it to your input data *X* and transform your input data with:

```
>>> import susi
>>> import matplotlib.pyplot as plt
>>> som = susi.SOMClustering()
>>> som.fit(X)
>>> bmu_list = som.get_bmus(X)
>>> plt.hist2d([x[0] for x in bmu_list], [x[1] for x in bmu_list]
```

get_clusters(*X*)
Calculate the SOM nodes on the unsupervised SOM grid per datapoint.

Parameters **X** (*np.ndarray*) – Input data

Returns List of SOM nodes, one for each input datapoint

Return type list of tuples (int, int)

get_datapoints_from_node(*node*)
Get all datapoints of one node.

Parameters **node** (*tuple, shape (int, int)*) – Node for which the linked datapoints are calculated

Returns **datapoints** – List of indices of the datapoints that are linked to *node*

Return type list of int

get_nbh_distance_weight_block(*nbh_func*, *bmus*)
Calculate distance weight matrix for all datapoints.

The combination of several distance weight matrices is called "block" in the following.

Parameters

- **neighborhood_func** (*float*) – Current neighborhood function
- **bmu_pos** (*tuple, shape=(int, int)*) – Position of calculated BMU of the current datapoint

Returns **dist_weight_block** – Neighborhood distance weight block between SOM and BMUs

Return type np.array of float, shape=(n_rows, n_columns)

get_nbh_distance_weight_matrix(*neighborhood_func*, *bmu_pos*)
Calculate neighborhood distance weight.

Parameters

- **neighborhood_func** (*float*) – Current neighborhood function
- **bmu_pos** (*tuple, shape=(int, int)*) – Position of calculated BMU of the current datapoint

Returns Neighborhood distance weight matrix between SOM and BMU

Return type np.array of float, shape=(n_rows, n_columns)

get_node_distance_matrix(*datapoint*, *som_array*)
Get distance of datapoint and node using Euclidean distance.

Parameters

- **datapoint** (*np.array, shape=(X.shape[1])*) – Datapoint = one row of the dataset *X*

- **som_array** (*np.array*) – Weight vectors of the SOM, shape = (self.n_rows, self.n_columns, X.shape[1])

Returns **distmat** – Distance between datapoint and each SOM node

Return type np.array of float

get_u_matrix(*mode='mean'*)
Calculate unified distance matrix (u-matrix).

Parameters **mode** (*str, optional (default="mean)*) – Choice of the averaging algorithm

Returns U-matrix containing the distances between all nodes of the unsupervised SOM. Shape = (n_rows*2-1, n_columns*2-1)

Return type np.array

Examples

Fit your SOM to input data *X* and then calculate the u-matrix with *get_u_matrix()*. You can plot the u-matrix then with e.g. *pyplot.imshow()*.

```
>>> import susi
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> som = susi.SOMClustering()
>>> som.fit(X)
>>> umat = som.get_u_matrix()
>>> plt.imshow(np.squeeze(umat))
```

get_u_mean(*nodelist*)
Calculate a mean value of the node entries in *nodelist*.

Parameters **nodelist** (*list of tuple (int, int)*) – List of nodes on the u-matrix containing distance values

Returns **u_mean** – Mean value

Return type *float*

init_unsuper_som()
Initialize map.

modify_weight_matrix_batch(*som_array*, *dist_weight_matrix*, *data*)
Modify weight matrix of the SOM for the online algorithm.

Parameters

- **som_array** (*np.array*) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **dist_weight_matrix** (*np.array of float*) – Current distance weight of the SOM for the specific node
- **data** (*np.array, optional*) – True vector(s)
- **learningrate** (*float*) – Current learning rate of the SOM

Returns Weight vector of the SOM after the modification

Return type np.array

set_bmus(*X*, *som_array=None*)
Set BMUs in the current SOM object.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The input samples.
- **som_array** (*np.array*) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])

train_unsupervised_som()
Train unsupervised SOM.

transform(*X*, *y=None*)
Transform input data.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The prediction input samples.
- **y** (*None, optional*) – Ignored.

Returns Predictions including the BMUs of each datapoint

Return type np.array of tuples (int, int)

Examples

Load the SOM, fit it to your input data *X* and transform your input data with:

```
>>> import susi
>>> som = susi.SOMClustering()
>>> som.fit(X)
>>> X_transformed = som.transform(X)
```

CHAPTER

SEVEN

SOMESTIMATOR

class susi.**SOMEstimator**(*n_rows: int = 10*, *n_columns: int = 10*, *init_mode_unsupervised: str = 'random'*, *init_mode_supervised: str = 'random'*, *n_iter_unsupervised: int = 1000*, *n_iter_supervised: int = 1000*, *train_mode_unsupervised: str = 'online'*, *train_mode_supervised: str = 'online'*, *neighborhood_mode_unsupervised: str = 'linear'*, *neighborhood_mode_supervised: str = 'linear'*, *learn_mode_unsupervised: str = 'min'*, *learn_mode_supervised: str = 'min'*, *distance_metric: str = 'euclidean'*, *learning_rate_start=0.5*, *learning_rate_end=0.05*, *nbh_dist_weight_mode: str = 'pseudo-gaussian'*, *missing_label_placeholder=None*, *n_jobs=None*, *random_state=None*, *verbose=0*)

Basic class for supervised self-organizing maps.

Parameters

- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **init_mode_unsupervised** (*str, optional (default="random")*) – Initialization mode of the unsupervised SOM
- **init_mode_supervised** (*str, optional (default="random")*) – Initialization mode of the supervised SOM
- **n_iter_unsupervised** (*int, optional (default=1000)*) – Number of iterations for the unsupervised SOM
- **n_iter_supervised** (*int, optional (default=1000)*) – Number of iterations for the supervised SOM
- **train_mode_unsupervised** (*str, optional (default="online")*) – Training mode of the unsupervised SOM
- **train_mode_supervised** (*str, optional (default="online")*) – Training mode of the supervised SOM
- **neighborhood_mode_unsupervised** (*str, optional (default="linear")*) – Neighborhood mode of the unsupervised SOM
- **neighborhood_mode_supervised** (*str, optional (default="linear")*) – Neighborhood mode of the supervised SOM
- **learn_mode_unsupervised** (*str, optional (default="min")*) – Learning mode of the unsupervised SOM
- **learn_mode_supervised** (*str, optional (default="min")*) – Learning mode of the supervised SOM

- **distance_metric** (*str, optional (default="euclidean")*) – Distance metric to compare on feature level (not SOM grid). Possible metrics: {"euclidean", "manhattan", "mahalanobis", "tanimoto"}. Note that "tanimoto" tends to be slow.
- **learning_rate_start** (*float, optional (default=0.5)*) – Learning rate start value
- **learning_rate_end** (*float, optional (default=0.05)*) – Learning rate end value (only needed for some lr definitions)
- **nbh_dist_weight_mode** (*str, optional (default="pseudo-gaussian")*) – Formula of the neighborhood distance weight. Possible formulas are: {"pseudo-gaussian", "mexican-hat"}.
- **missing_label_placeholder** (*int or str or None, optional (default=None)*) – Label placeholder for datapoints with no label. This is needed for semi-supervised learning.
- **n_jobs** (*int or None, optional (default=None)*) – The number of jobs to run in parallel.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
- **verbose** (*int, optional (default=0)*) – Controls the verbosity.

Variables

- **node_list_** (*np.array of (int, int) tuples*) – List of 2-dimensional coordinates of SOM nodes
- **radius_max_** (*float, int*) – Maximum radius of the neighborhood function
- **radius_min_** (*float, int*) – Minimum radius of the neighborhood function
- **unsuper_som_** (*np.array*) – Weight vectors of the unsupervised SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **X_** (*np.array*) – Input data
- **fitted_** (*bool*) – States if estimator is fitted to X
- **max_iterations_** (*int*) – Maximum number of iterations for the current training
- **bmus_** (*list of (int, int) tuples*) – List of best matching units (BMUs) of the dataset X
- **sample_weights_** (*TODO*) –
- **n_features_in_** (*int*) – Number of input features

calc_estimation_output (*datapoint*, *mode='bmu'*)

Get SOM output for fixed SOM.

The given datapoint doesn't have to belong to the training set of the input SOM.

Parameters

- **datapoint** (*np.array, shape=(X.shape[1])*) – Datapoint = one row of the dataset X
- **mode** (*str, optional (default="bmu")*) – Mode of the regression output calculation

Returns

- **estimation_output** (*object*) – Content of SOM node which is linked to the datapoint. Classification: the label. Regression: the target variable.
- *TODO Implement handling of incomplete datapoints*

- *TODO implement "neighborhood" mode*

fit (*X*, *y=None*)

Fit supervised SOM to the input data.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The prediction input samples.
- **y** (*array-like matrix of shape = [n_samples, 1]*) – The labels (ground truth) of the input samples

Returns self

Return type object

Examples

Load the SOM and fit it to your input data *X* and the labels *y* with:

```
>>> import susi
>>> som = susi.SOMRegressor()
>>> som.fit(X, y)
```

fit_estimator (*X*, *y*)

Fit supervised SOM to the (checked) input data.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The prediction input samples.
- **y** (*array-like matrix of shape = [n_samples, 1]*) – The labels (ground truth) of the input samples

fit_transform (*X*, *y=None*)

Fit to the input data and transform it.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The training and prediction input samples.
- **y** (*array-like matrix of shape = [n_samples, 1]*) – The labels (ground truth) of the input samples

Returns Predictions including the BMUs of each datapoint

Return type np.array of tuples (int, int)

Examples

Load the SOM, fit it to your input data *X* and transform your input data with:

```
>>> import susi
>>> som = susi.SOMClassifier()
>>> tuples = som.fit_transform(X, y)
```

get_estimation_map ()

Return SOM grid with the estimated value on each node.

Examples

Fit the SOM on your data *X, y*:

```
>>> import susi
>>> import matplotlib.pyplot as plt
>>> som = susi.SOMClassifier()
>>> som.fit(X, y)
>>> estimation_map = som.get_estimation_map()
>>> plt.imshow(np.squeeze(estimation_map,) cmap="viridis_r")
```

get_random_datapoint()
Find and return random datapoint from labeled dataset.

abstract init_super_som()
Initialize map.

modify_weight_matrix_supervised(*dist_weight_matrix*, *true_vector=None*, *learningrate=None*)
Modify weights of the supervised SOM, either online or batch.

Parameters

- **som_array** (*np.array*) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **dist_weight_matrix** (*np.array of float*) – Current distance weight of the SOM for the specific node
- **data** (*np.array, optional*) – True vector(s)
- **learningrate** (*float, optional*) – Current learning rate of the SOM

Returns **modify_weight_matrix** – Weight vector of the SOM after the modification

Return type np.array

predict(*X*, *y=None*)
Predict output of data X.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The prediction input samples.
- **y** (*None, optional*) – Ignored.

Returns **y_pred** – List of predicted values.

Return type list of float

Examples

Fit the SOM on your data *X, y*:

```
>>> import susi
>>> som = susi.SOMClassifier()
>>> som.fit(X, y)
>>> y_pred = som.predict(X)
```

train_supervised_som()
Train supervised SOM.

CHAPTER

EIGHT

SOMREGRESSOR

class susi.**SOMRegressor**(*n_rows: int = 10*, *n_columns: int = 10*, *init_mode_unsupervised: str = 'random'*, *init_mode_supervised: str = 'random'*, *n_iter_unsupervised: int = 1000*, *n_iter_supervised: int = 1000*, *train_mode_unsupervised: str = 'online'*, *train_mode_supervised: str = 'online'*, *neighborhood_mode_unsupervised: str = 'linear'*, *neighborhood_mode_supervised: str = 'linear'*, *learn_mode_unsupervised: str = 'min'*, *learn_mode_supervised: str = 'min'*, *distance_metric: str = 'euclidean'*, *learning_rate_start=0.5*, *learning_rate_end=0.05*, *nbh_dist_weight_mode: str = 'pseudo-gaussian'*, *missing_label_placeholder=None*, *n_jobs=None*, *random_state=None*, *verbose=0*)

Supervised SOM for estimating continuous variables (= regression).

Parameters

- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **init_mode_unsupervised** (*str, optional (default="random")*) – Initialization mode of the unsupervised SOM
- **init_mode_supervised** (*str, optional (default="random")*) – Initialization mode of the supervised SOM
- **n_iter_unsupervised** (*int, optional (default=1000)*) – Number of iterations for the unsupervised SOM
- **n_iter_supervised** (*int, optional (default=1000)*) – Number of iterations for the supervised SOM
- **train_mode_unsupervised** (*str, optional (default="online")*) – Training mode of the unsupervised SOM
- **train_mode_supervised** (*str, optional (default="online")*) – Training mode of the supervised SOM
- **neighborhood_mode_unsupervised** (*str, optional (default="linear")*) – Neighborhood mode of the unsupervised SOM
- **neighborhood_mode_supervised** (*str, optional (default="linear")*) – Neighborhood mode of the supervised SOM
- **learn_mode_unsupervised** (*str, optional (default="min")*) – Learning mode of the unsupervised SOM
- **learn_mode_supervised** (*str, optional (default="min")*) – Learning mode of the supervised SOM

- **distance_metric** (*str, optional (default="euclidean")*) – Distance metric to compare on feature level (not SOM grid). Possible metrics: {"euclidean", "manhattan", "mahalanobis", "tanimoto"}. Note that "tanimoto" tends to be slow.
- **learning_rate_start** (*float, optional (default=0.5)*) – Learning rate start value
- **learning_rate_end** (*float, optional (default=0.05)*) – Learning rate end value (only needed for some lr definitions)
- **nbh_dist_weight_mode** (*str, optional (default="pseudo-gaussian")*) – Formula of the neighborhood distance weight. Possible formulas are: {"pseudo-gaussian", "mexican-hat"}.
- **missing_label_placeholder** (*int or str or None, optional (default=None)*) – Label placeholder for datapoints with no label. This is needed for semi-supervised learning.
- **n_jobs** (*int or None, optional (default=None)*) – The number of jobs to run in parallel.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
- **verbose** (*int, optional (default=0)*) – Controls the verbosity.

Variables

- **node_list_** (*np.array of (int, int) tuples*) – List of 2-dimensional coordinates of SOM nodes
- **radius_max_** (*float, int*) – Maximum radius of the neighborhood function
- **radius_min_** (*float, int*) – Minimum radius of the neighborhood function
- **unsuper_som_** (*np.array*) – Weight vectors of the unsupervised SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **X_** (*np.array*) – Input data
- **fitted_** (*bool*) – States if estimator is fitted to X
- **max_iterations_** (*int*) – Maximum number of iterations for the current training
- **bmus_** (*list of (int, int) tuples*) – List of best matching units (BMUs) of the dataset X
- **sample_weights_** (*TODO*) –
- **n_regression_vars_** (*int*) – Number of regression variables. In most examples, this equals one.
- **n_features_in_** (*int*) – Number of input features

init_super_som()

Initialize map for regression.

CHAPTER

NINE

SOMCLASSIFIER

class susi.**SOMClassifier**(*n_rows: int = 10*, *n_columns: int = 10*, *init_mode_unsupervised: str = 'random'*, *init_mode_supervised: str = 'majority'*, *n_iter_unsupervised: int = 1000*, *n_iter_supervised: int = 1000*, *train_mode_unsupervised: str = 'online'*, *train_mode_supervised: str = 'online'*, *neighborhood_mode_unsupervised: str = 'linear'*, *neighborhood_mode_supervised: str = 'linear'*, *learn_mode_unsupervised: str = 'min'*, *learn_mode_supervised: str = 'min'*, *distance_metric: str = 'euclidean'*, *learning_rate_start=0.5*, *learning_rate_end=0.05*, *nbh_dist_weight_mode: str = 'pseudo-gaussian'*, *missing_label_placeholder=None*, *do_class_weighting=True*, *n_jobs=None*, *random_state=None*, *verbose=0*)

Supervised SOM for estimating discrete variables (= classification).

Parameters

- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **init_mode_unsupervised** (*str, optional (default="random")*) – Initialization mode of the unsupervised SOM
- **init_mode_supervised** (*str, optional (default="majority")*) – Initialization mode of the classification SOM
- **n_iter_unsupervised** (*int, optional (default=1000)*) – Number of iterations for the unsupervised SOM
- **n_iter_supervised** (*int, optional (default=1000)*) – Number of iterations for the classification SOM
- **train_mode_unsupervised** (*str, optional (default="online")*) – Training mode of the unsupervised SOM
- **train_mode_supervised** (*str, optional (default="online")*) – Training mode of the classification SOM
- **neighborhood_mode_unsupervised** (*str, optional (default="linear")*) – Neighborhood mode of the unsupervised SOM
- **neighborhood_mode_supervised** (*str, optional (default="linear")*) – Neighborhood mode of the classification SOM
- **learn_mode_unsupervised** (*str, optional (default="min")*) – Learning mode of the unsupervised SOM
- **learn_mode_supervised** (*str, optional (default="min")*) – Learning mode of the classification SOM

- **distance_metric** (*str, optional (default="euclidean")*) – Distance metric to compare on feature level (not SOM grid). Possible metrics: {"euclidean", "manhattan", "mahalanobis", "tanimoto"}. Note that "tanimoto" tends to be slow.
- **learning_rate_start** (*float, optional (default=0.5)*) – Learning rate start value
- **learning_rate_end** (*float, optional (default=0.05)*) – Learning rate end value (only needed for some lr definitions)
- **nbh_dist_weight_mode** (*str, optional (default="pseudo-gaussian")*) – Formula of the neighborhood distance weight. Possible formulas are: {"pseudo-gaussian", "mexican-hat"}.
- **missing_label_placeholder** (*int or str or None, optional (default=None)*) – Label placeholder for datapoints with no label. This is needed for semi-supervised learning.
- **do_class_weighting** (*bool, optional (default=True)*) – If true, classes are weighted.
- **n_jobs** (*int or None, optional (default=None)*) – The number of jobs to run in parallel.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
- **verbose** (*int, optional (default=0)*) – Controls the verbosity.

Variables

- **node_list_** (*np.array of (int, int) tuples*) – List of 2-dimensional coordinates of SOM nodes
- **radius_max_** (*float, int*) – Maximum radius of the neighborhood function
- **radius_min_** (*float, int*) – Minimum radius of the neighborhood function
- **unsuper_som_** (*np.array*) – Weight vectors of the unsupervised SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **X_** (*np.array*) – Input data
- **fitted_** (*bool*) – States if estimator is fitted to X
- **max_iterations_** (*int*) – Maximum number of iterations for the current training
- **bmus_** (*list of (int, int) tuples*) – List of best matching units (BMUs) of the dataset X
- **placeholder_dict_** (*dict*) – Dict of placeholders for initializing nodes without mapped class.
- **n_features_in_** (*int*) – Number of input features

change_class_proba(*learningrate*, *dist_weight_matrix*, *class_weight*)

Calculate probability of changing class in a node.

Parameters

- **learningrate** (*float*) – Current learning rate of the SOM
- **dist_weight_matrix** (*np.array of float*) – Current distance weight of the SOM for the specific node
- **class_weight** (*float*) – Weight of the class of the current datapoint

Returns **change_class_bool** – Matrix with one boolean for each node on the SOM node. If true, the value of the respective SOM node gets changed. If false, the value of the respective SOM node stays the same.

Return type np.array, shape = (n_rows, n_columns)

fit (*X*, *y=None*)
Fit classification SOM to the input data.

Parameters

- **X** (*array-like matrix of shape = [n_samples, n_features]*) – The prediction input samples.
- **y** (*array-like matrix of shape = [n_samples, 1]*) – The labels (ground truth) of the input samples

Returns **self**

Return type object

Examples

Load the SOM and fit it to your input data *X* and the labels *y* with:

```
>>> import susi
>>> som = susi.SOMClassifier()
>>> som.fit(X, y)
```

init_super_som ()
Initialize map.

modify_weight_matrix_supervised (*dist_weight_matrix*, *true_vector=None*, *learningrate=None*)
Modify weight matrix of the SOM.

Parameters

- **dist_weight_matrix** (*np.array of float*) – Current distance weight of the SOM for the specific node
- **learningrate** (*float, optional*) – Current learning rate of the SOM
- **true_vector** (*np.array*) – Datapoint = one row of the dataset X

Returns **new_matrix** – Weight vector of the SOM after the modification

Return type np.array

set_placeholder ()
Set placeholder depending on the class dtype.

CHAPTER

TEN

SOMESTIMATOR

SOMPlots functions.

susi.SOMPlots.**plot_estimation_map**(*estimation_map*, *cbar_label='Variable in unit'*, *cmap='viridis'*, *fontsize=20*)

[summary]

Parameters

- **estimation_map** (*np.array*) – Estimation map of the size (n_rows, n_columns)
- **cbar_label** (*str, optional*) – Label of the colorbar, by default "Variable in unit"
- **cmap** (*str, optional (default="viridis")*) – Colormap
- **fontsize** (*int, optional (default=20)*) – Fontsize of the labels

Returns **ax** – Plot axis

Return type pyplot.axis

susi.SOMPlots.**plot_som_histogram**(*bmu_list*, *n_rows*, *n_columns*, *n_datapoints_cbar=5*, *fontsize=22*)

Plot 2D Histogram of SOM.

Plot 2D Histogram with one bin for each SOM node. The content of one bin is the number of datapoints matched to the specific node.

Parameters

- **bmu_list** (*list of (int, int) tuples*) – Position of best matching units (row, column) for each datapoint
- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **n_datapoints_cbar** (*int, optional (default=5)*) – Maximum number of datapoints shown on the colorbar
- **fontsize** (*int, optional (default=22)*) – Fontsize of the labels

Returns **ax** – Plot axis

Return type pyplot.axis

susi.SOMPlots.**plot_umatrix**(*u_matrix*, *n_rows*, *n_colums*, *cmap='Greys'*, *fontsize=18*)

Plot u-matrix.

Parameters

- **u_matrix** (*np.array*) – U-matrix containing the distances between all nodes of the unsupervised SOM. Shape = (n_rows*2-1, n_columns*2-1)
- **n_rows** (*int, optional (default=10)*) – Number of rows for the SOM grid
- **n_columns** (*int, optional (default=10)*) – Number of columns for the SOM grid
- **cmap** (*str, optional (default="Greys)*) – Colormap
- **fontsize** (*int, optional (default=18)*) – Fontsize of the labels

Returns **ax** – Plot axis

Return type pyplot.axis

CHAPTER

ELEVEN

SOMUTILS

SOMUtils functions.

susi.SOMUtils.**check_estimation_input**(*X*, *y*, *is_classification=False*)
Check input arrays.

This function is adapted from sklearn.utils.validation.

Parameters

- **X** (*nd-array or list*) – Input data.
- **y** (*nd-array, list*) – Labels.
- **is_classification** (boolean (default=`False`)) – Wether the data is used for classification or regression tasks.

Returns

- **X** (*object*) – The converted and validated *X*.
- **y** (*object*) – The converted and validated *y*.

susi.SOMUtils.**decreasing_rate**(*a_1*, *a_2*, *iteration_max*, *iteration*, *mode*)
Return a decreasing rate from collection.

Parameters

- **a_1** (*float*) – Starting value of decreasing rate
- **a_2** (*float*) – End value of decreasing rate
- **iteration_max** (*int*) – Maximum number of iterations
- **iteration** (*int*) – Current number of iterations
- **mode** (*str*) – Mode (= formula) of the decreasing rate

Returns **rate** – Decreasing rate

Return type float

Examples

```
>>> import susi
>>> susi.decreasing_rate(0.8, 0.1, 100, 5, "exp")
```

susi.SOMUtils.**modify_weight_matrix_online**(*som_array*, *dist_weight_matrix*, *true_vector*, *learningrate*)

Modify weight matrix of the SOM for the online algorithm.

Parameters

- **som_array** (*np.array*) – Weight vectors of the SOM shape = (self.n_rows, self.n_columns, X.shape[1])
- **dist_weight_matrix** (*np.array of float*) – Current distance weight of the SOM for the specific node
- **true_vector** (*np.array*) – True vector
- **learningrate** (*float*) – Current learning rate of the SOM

Returns Weight vector of the SOM after the modification

Return type np.array

BIBLIOGRAPHY

[RieseEtAl2020] F. M. Riese, S. Keller and S. Hinz, “Supervised and Semi-Supervised Self-Organizing Maps for Regression and Classification Focusing on Hyperspectral Data”, *Remote Sensing*, vol. 12, no. 1, 2020. Link

PYTHON MODULE INDEX

S

INDEX

T