2023WS - 188.413 Self-Organizing Systems - Darovskikh Leonid (00828589), Hunner Markus (01503441), Loidolt Lukas (01634039)

The implementations, as well as this notebook can be found online at:
https://github.com/markhun/2023W-SOS-A3

Assignment 3: LabelSOM Visualiztion

Before running this notebook, be sure to executed `git submodule update --init --recursive` to checkout `PySOMVis`.

Then execute `make dev` to generate a Python `venv` and with all necessary dependencies.

And last but not least execute `make install` to install `PySOMVis` and `labelsom` from within this repository into the generated `venv`.

See README.md for further details.

Datasets and SOMs used

We used the `chainlink` and `10clusters` datasets as provided by `PySOMVis` to train a small (10x10) and large (100x60) SOM via the Java SOMToolbox on each of the datasets.

In [1]:
```
from pathlib import Path

chainlink_10x10_data = Path("./trainedData/chainlink/10x10")
chainlink_100x60_data = Path("./trainedData/chainlink/100x60/")
_10clusters_10x10_data = Path("./trainedData/10clusters/10x10")
_10clusters_100x60_data = Path("./trainedData/10clusters/100x60/")
```

In [2]:
```
from PySOMVis.SOMToolBox_Parse import SOMToolBox_Parse


def read_in_trained_SOM(som_directory: Path):
    """Helper function to read in necessary SOM data via `PySOMVis.SOMToolBox_Parse`"""
    if not som_directory.is_dir():
        raise RuntimeError("`som_directory` must be a directory.")
    files = [f for f in som_directory.iterdir() if f.is_file()]

    components, input_data, weights = None, None, None
    for file_ in files:
        if file_.suffixes == [".tv"]:
            components = SOMToolBox_Parse(str(file_)).read_weight_file()
        elif file_.suffixes == [".vec"]:
            input_data = SOMToolBox_Parse(str(file_)).read_weight_file()
        elif file_.suffixes == [".wgt", ".gz"]:
            weights = SOMToolBox_Parse(str(file_)).read_weight_file()

    if components is None:
        raise RuntimeError("No *.tv file found in directory.")
    if input_data is None:
        raise RuntimeError("No *.vec file found in directory.")
    if weights is None:
        raise RuntimeError("No *.wgt.gz file found in directory.")

    m, n = weights["ydim"], weights["xdim"]
    attribute_names = list(components["arr"][:, 1])
    if not input_data["vec_dim"] == len(attribute_names):
        raise RuntimeError(
            "input data vector dimensions don't match number of attributes from components."
        )
    weights, input_data = weights["arr"], input_data["arr"]

    return m, n, attribute_names, weights, input_data
```

Visualizing 10clusters via PySOMVis

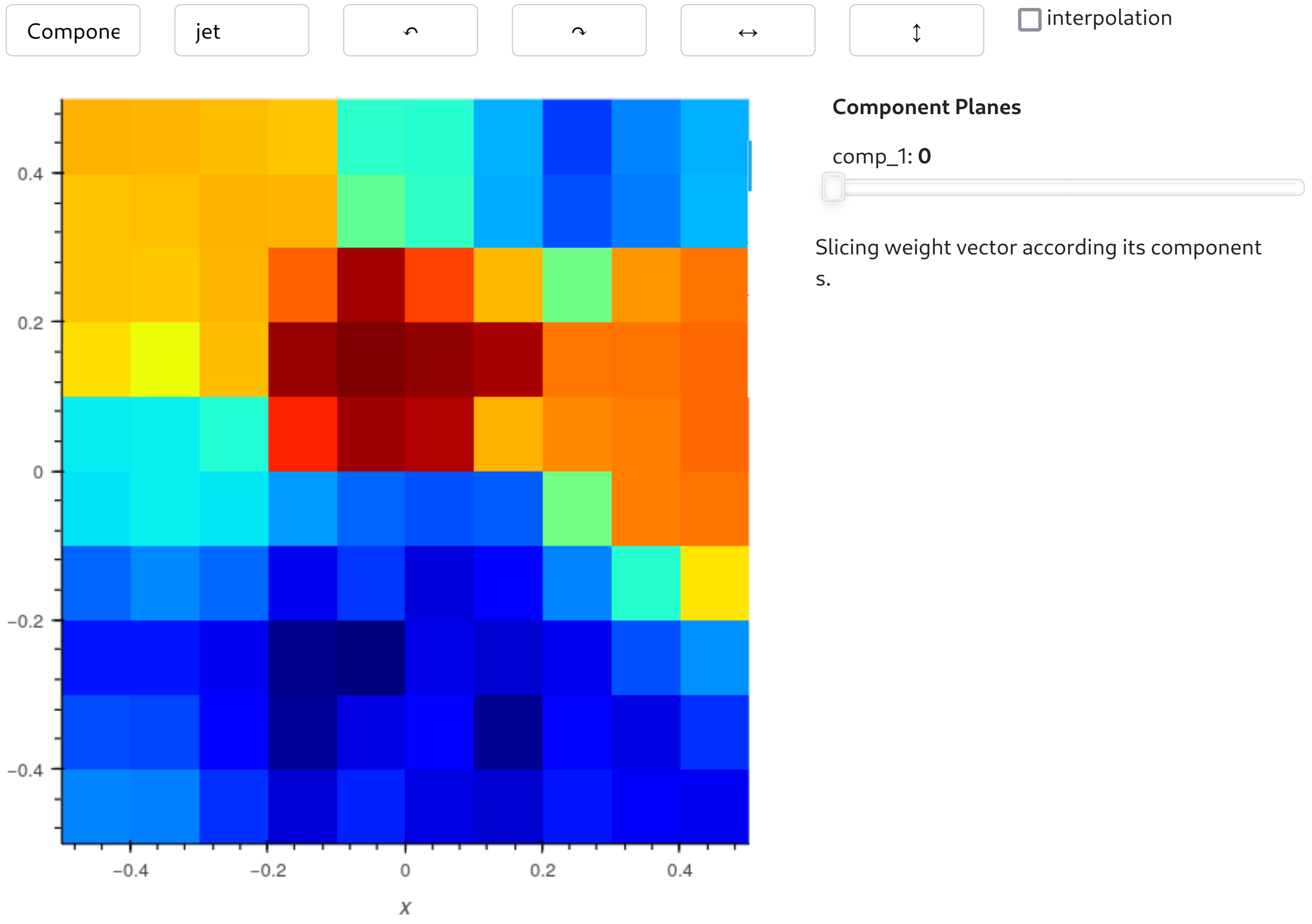
In [3]:
```
from PySOMVis.SOMToolBox_Parse import SOMToolBox_Parse

_10cluster_class_info = SOMToolBox_Parse("./PySOMVis/PySOMVis/datasets/10clusters/10clusters.cls").read_weight_file()
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/10clusters/10x10"))
```

In [4]:
```
from PySOMVis.pysomvis import PySOMVis

vis = PySOMVis(weights=weights,
               m=m,
               n=n,
               dimension=len(attribute_names),
               input_data=input_data,
               classes_names=_10cluster_class_info['classes_names'],
               classes=_10cluster_class_info['arr'][:,1],
               component_names=attribute_names,
              )
vis._mainview
```

Out[4]:



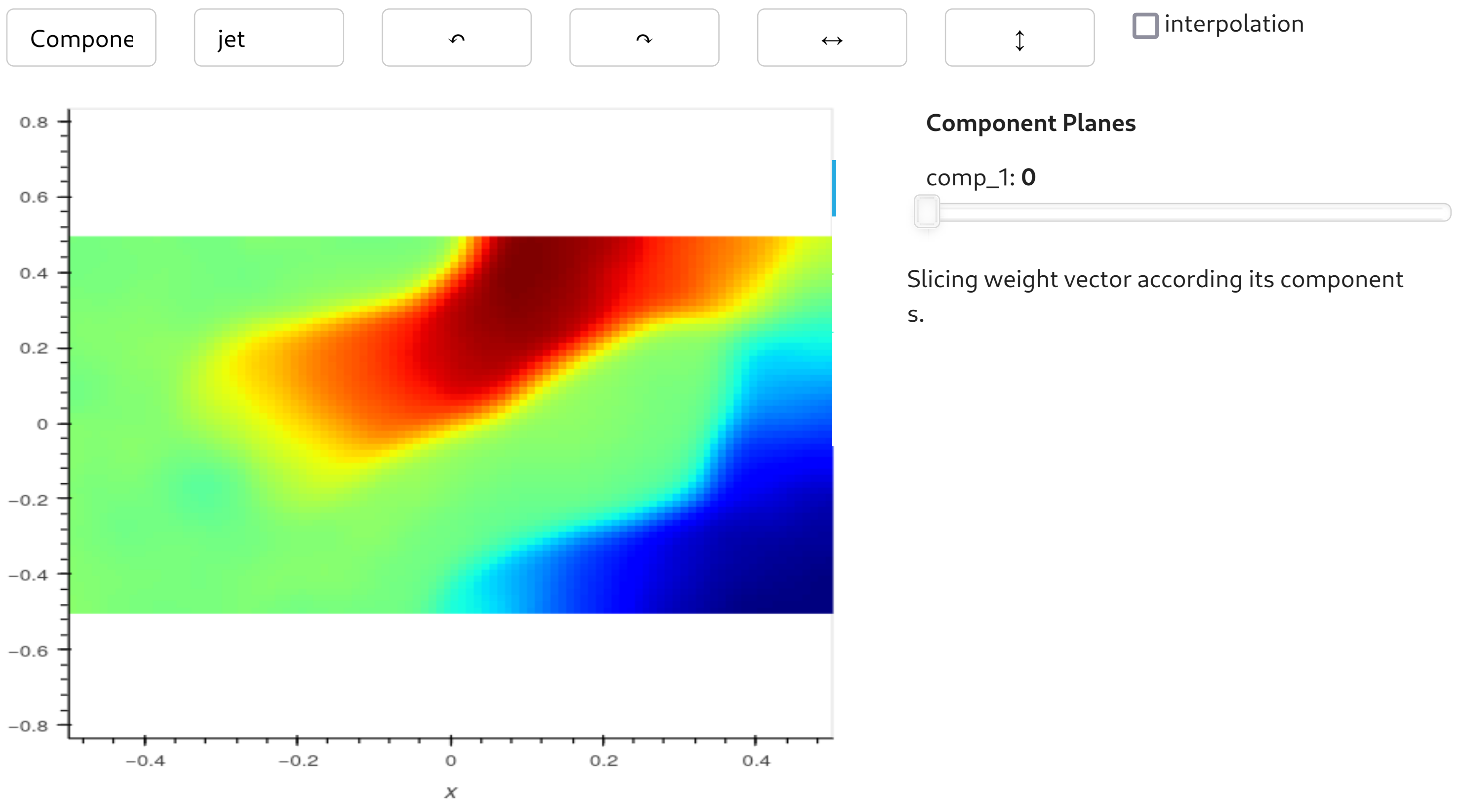
Slicing weight vector according its component s.

Visualizing chainlink via PySOMVis

In [5]:
```
chainlink_class_info = SOMToolBox_Parse("./PySOMVis/PySOMVis/datasets/chainlink/chainlink.cls").read_weight_file()
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/chainlink/100x60"))
```

In [6]:
```
vis = PySOMVis(weights=weights,
               m=m,
               n=n,
               dimension=len(attribute_names),
               input_data=input_data,
               classes_names=chainlink_class_info['classes_names'],
               classes=chainlink_class_info['arr'][:,1],
               component_names=attribute_names,
              )
vis._mainview
```

Out[6]:



Slicing weight vector according its component s.

Using `labelsom` to visualize chainlink

We re-implemented the LabelSOM visualization as defined in `SOMToolBox/src/core/at/tuwien/ifs/somtoolbox/output/labeling/LabelSOM.java` . The implementation itself can be found within the `labelsom` Python library within this repository.

Similar to the visualization provided by the SOMToolBox our `labelsom` library visualizes a SOM not only by labeling its units but also by combining the resulting table with a hit histogram coloring.

1. Read in data via `PySOMVis.SOMToolBox_Parse`

In [7]:
```
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/chainlink/10x10"))
```

2. Use `labelsom` to generate a label matrix and hit histogram matrix

We generate a table containing our labeling result (`label_matrix`) together with a hit histogram (`hit_histogram`). The implementation for generating the hit histogram, as well as for finding units mapped onto other units is inspired by the `HitHist` function within `PySOMVis/PySOMVis/coding_assignment.ipynb` . Its implementation can be found in `labelsom.generate_unit_idx_to_mapped_indices_mapping` .

In [8]:
```
import numpy as np

import labelsom

label_matrix, hit_histogram = labelsom.generate_label_matrix_and_hit_histogram(
    m=m, n=n, weights=weights, input_data=input_data, attribute_names=attribute_names,
    number_of_labels_to_generate=3,
    ignore_labels_with_zero=False,
)

with np.printoptions(threshold=5, linewidth= 300):
    print(label_matrix)
```

```
[[list([comp_2, comp_1, comp_3]) list([comp_2, comp_1, comp_3]) list([comp_2, comp_1, comp_3]) ... list([comp_2, comp_1, c
omp_3]) list([comp_2, comp_1, comp_3]) list([comp_2, comp_1, comp_3])]
 [list([comp_2, comp_1, comp_3]) list([comp_2, comp_1, comp_3]) list([comp_2, comp_1, comp_3]) ... list([comp_2, comp_1, c
omp_3]) list([comp_2, comp_1, comp_3]) list([comp_1, comp_2, comp_3])]
 [list([comp_2, comp_1, comp_3]) list([]) list([]) ... list([]) list([comp_1, comp_2, comp_3]) list([comp_1, comp_2, comp_
3])]
 ...
 [list([comp_2, comp_3, comp_1]) list([comp_2, comp_3, comp_1]) list([]) ... list([]) list([comp_3, comp_1, comp_2]) list
([comp_3, comp_1, comp_2])]
 [list([comp_2, comp_3, comp_1]) list([comp_2, comp_3, comp_1]) list([comp_2, comp_3, comp_1]) ... list([comp_3, comp_1, c
omp_2]) list([comp_3, comp_1, comp_2]) list([comp_3, comp_1, comp_2])]
 [list([comp_2, comp_3, comp_1]) list([comp_2, comp_3, comp_1]) list([comp_2, comp_3, comp_1]) ... list([comp_3, comp_1, c
omp_2]) list([comp_3, comp_1, comp_2]) list([comp_3, comp_1, comp_2])]]
```

3. Use `labelsom` to pretty print the `label_matrix`

The resulting labeling table and hit histogram data can be visualized in the form of an HTML table. The caller may choose if the visualization shall include the *mean* and *quantization error* values for each label.

In [9]:
```
labelsom.pretty_print_label_matrix(
    label_matrix,
    hit_histogram,
    include_mean=True,  # wether to include the mean for each label
    include_quantization_error=True,  # wether to include the qe for each label
)
```

```
Out[9]:                      0                           1                           2                           3
   0   comp_2  m:1.73   qe:0.02    comp_2  m:1.48   qe:0.04    comp_2  m:1.17   qe:0.04    comp_2  m:1.04   qe:0.03    comp_1  m:0.93   qe:0.04
       comp_1  m:0.67   qe:0.03    comp_1  m:0.83   qe:0.04    comp_1  m:0.97   qe:0.04    comp_1  m:0.98   qe:0.04    comp_2  m:0.70   qe:0.04
       comp_3  m:-0.03  qe:0.03    comp_3  m:-0.00  qe:0.06    comp_3  m:-0.00  qe:0.04    comp_3  m:0.02   qe:0.04    comp_3  m:0.03   qe:0.04

   1   comp_2  m:1.86   qe:0.04    comp_2  m:1.64   qe:0.04    comp_2  m:1.38   qe:0.04    comp_1  m:0.99   qe:0.03    comp_1  m:0.90   qe:0.03
       comp_1  m:0.52   qe:0.05    comp_1  m:0.72   qe:0.04    comp_1  m:0.96   qe:0.04    comp_2  m:0.89   qe:0.03    comp_2  m:0.53   qe:0.05
       comp_3  m:-0.00  qe:0.04    comp_3  m:0.01   qe:0.04    comp_3  m:0.02   qe:0.05    comp_3  m:0.00   qe:0.04    comp_3  m:-0.03  qe:0.04

   2   comp_2  m:1.96   qe:0.05                                                                                        comp_1  m:0.69   qe:0.02
       comp_1  m:0.24   qe:0.06                                                                                        comp_2  m:0.35   qe:0.03
       comp_3  m:-0.01  qe:0.03                                                                                        comp_3  m:-0.01  qe:0.03

   3   comp_2  m:2.01   qe:0.03                                comp_2  m:0.93   qe:0.03    comp_2  m:0.85   qe:0.04
       comp_3  m:0.02   qe:0.04                                comp_1  m:0.00   qe:0.05    comp_1  m:-0.01  qe:0.03
       comp_1  m:0.01   qe:0.07                                comp_3  m:-0.37  qe:0.05    comp_3  m:-0.56  qe:0.06

   4   comp_2  m:1.96   qe:0.04                                comp_2  m:1.03   qe:0.03    comp_2  m:0.95   qe:0.03
       comp_3  m:-0.01  qe:0.04                                comp_1  m:-0.03  qe:0.04    comp_1  m:0.01   qe:0.03
       comp_1  m:-0.30  qe:0.05                                comp_3  m:-0.23  qe:0.05    comp_3  m:-0.16  qe:0.04

   5   comp_2  m:1.86   qe:0.04                                comp_2  m:0.99   qe:0.03    comp_2  m:1.01   qe:0.04    comp_2  m:0.87   qe:0.04
       comp_3  m:-0.00  qe:0.04                                comp_3  m:0.04   qe:0.04    comp_3  m:0.18   qe:0.04    comp_3  m:0.36   qe:0.03
       comp_1  m:-0.50  qe:0.06                                comp_1  m:-0.01  qe:0.04    comp_1  m:0.02   qe:0.02    comp_1  m:0.02   qe:0.03
```

	0	1	2	3	
6	comp_2 m:1.72 qe:0.04 comp_3 m:-0.02 qe:0.05 comp_1 m:-0.70 qe:0.04	comp_2 m:1.58 qe:0.04 comp_3 m:-0.01 qe:0.03 comp_1 m:-0.79 qe:0.04		comp_2 m:0.95 qe:0.03 comp_3 m:0.45 qe:0.03 comp_1 m:-0.02 qe:0.03	comp_2 m:0.80 qe:0.04 comp_3 m:0.60 qe:0.05 comp_1 m:0.01 qe:0.04
7	comp_2 m:1.46 qe:0.03 comp_3 m:0.01 qe:0.04 comp_1 m:-0.91 qe:0.04	comp_2 m:1.33 qe:0.03 comp_3 m:0.02 qe:0.04 comp_1 m:-0.94 qe:0.03		comp_2 m:0.80 qe:0.02 comp_3 m:0.49 qe:0.03 comp_1 m:-0.04 qe:0.02	comp_3 m:0.74 qe:0.05 comp_2 m:0.66 qe:0.04 comp_1 m:0.02 qe:0.04
8	comp_2 m:1.12 qe:0.04 comp_3 m:0.02 qe:0.05 comp_1 m:-1.00 qe:0.03	comp_2 m:0.98 qe:0.02 comp_3 m:-0.00 qe:0.04 comp_1 m:-0.98 qe:0.02	comp_2 m:0.63 qe:0.05 comp_3 m:-0.01 qe:0.03 comp_1 m:-0.91 qe:0.03	comp_2 m:0.39 qe:0.02 comp_3 m:0.05 qe:0.05 comp_1 m:-0.69 qe:0.02	
9	comp_2 m:0.88 qe:0.02 comp_3 m:-0.02 qe:0.02 comp_1 m:-0.98 qe:0.04	comp_2 m:0.75 qe:0.03 comp_3 m:0.01 qe:0.04 comp_1 m:-0.98 qe:0.03	comp_2 m:0.47 qe:0.05 comp_3 m:0.01 qe:0.04 comp_1 m:-0.84 qe:0.04	comp_2 m:0.29 qe:0.03 comp_3 m:-0.01 qe:0.04 comp_1 m:-0.68 qe:0.05	comp_2 m:0.15 qe:0.05 comp_3 m:-0.02 qe:0.04 comp_1 m:-0.47 qe:0.06

Instead of visualizing our labelsom implementation in-line within a Python notebook the result can also be dumped into an HTML file:

```
In [10]: labelsom.write_labelsom_to_file(
             label_matrix,
             hit_histogram,
             include_mean=True,  # wether to include the mean for each label
             include_quantization_error=True,  # wether to include the qe for each label
             directory_to_write_file_to=Path("./generated_visualizations"),
             file_name="chainlink_10x10",
         )
```

```
Out[10]: PosixPath('generated_visualizations/chainlink_10x10.html')
```

4. Parameters for LabelSOM generation and table visualization

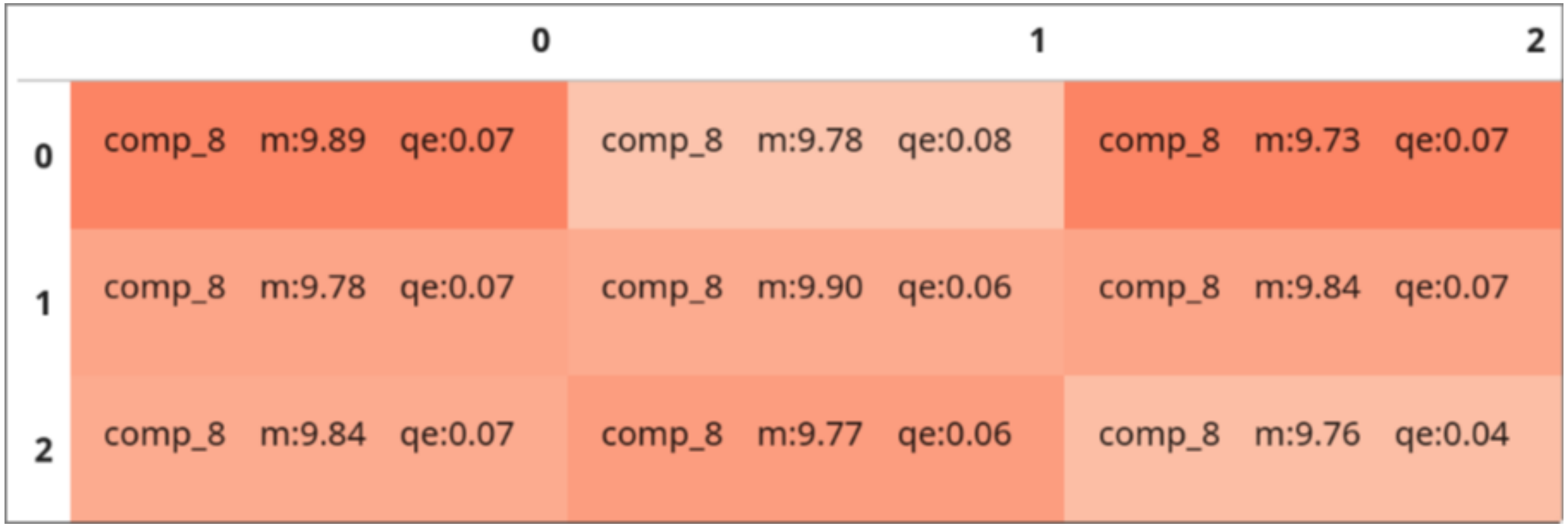
Several parameters enable tweaking our LabelSOM implementation as well as its visual table representation. They are detailed in the sections below.

Determine number of labels to generate

Like the Java SOMToolbox, our implementation allows users to change the number of labels assigned to each SOM unit through the `number_of_labels_to_generate` parameter passed to `generate_label_matrix_and_hit_histogram` .

If this parameter is set to 1, each unit will be labelled only with the feature most 'relevant' in its overall contribution to the unit's weight vector, i.e., the feature with the highest mean and lowest quantization error among all input vectors mapped to the unit. Setting it to a higher number will include less relevant features as well, in descending order of importance.

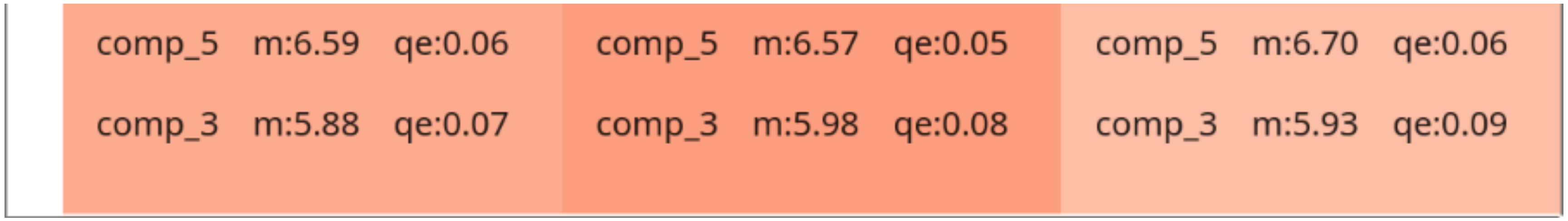
The following images show parts of the LabelSOM matrix generated from the '10clusters' dataset with `number_of_labels_to_generate` set to 1 and 5, respectively.



	0	1	2
0	comp_8 m:9.89 qe:0.07	comp_8 m:9.78 qe:0.08	comp_8 m:9.73 qe:0.07
1	comp_8 m:9.78 qe:0.07	comp_8 m:9.90 qe:0.06	comp_8 m:9.84 qe:0.07
2	comp_8 m:9.84 qe:0.07	comp_8 m:9.77 qe:0.06	comp_8 m:9.76 qe:0.04

10clusters LabelSOM with only 1 label

	0	1	2
0	comp_8 m:9.89 qe:0.07 comp_6 m:8.63 qe:0.07 comp_9 m:7.95 qe:0.07 comp_5 m:6.54 qe:0.05 comp_3 m:5.84 qe:0.07	comp_8 m:9.78 qe:0.08 comp_6 m:8.58 qe:0.09 comp_9 m:8.00 qe:0.08 comp_5 m:6.60 qe:0.07 comp_3 m:5.95 qe:0.05	comp_8 m:9.73 qe:0.07 comp_6 m:8.70 qe:0.07 comp_9 m:7.93 qe:0.08 comp_5 m:6.50 qe:0.06 comp_3 m:5.99 qe:0.08
1	comp_8 m:9.78 qe:0.07 comp_6 m:8.60 qe:0.07 comp_9 m:7.91 qe:0.07 comp_5 m:6.52 qe:0.05 comp_3 m:5.87 qe:0.07	comp_8 m:9.90 qe:0.06 comp_6 m:8.68 qe:0.07 comp_9 m:7.97 qe:0.08 comp_5 m:6.58 qe:0.06 comp_3 m:6.03 qe:0.05	comp_8 m:9.84 qe:0.07 comp_6 m:8.67 qe:0.09 comp_9 m:7.89 qe:0.06 comp_5 m:6.69 qe:0.06 comp_3 m:5.98 qe:0.06
2	comp_8 m:9.84 qe:0.07 comp_6 m:8.59 qe:0.07 comp_9 m:7.86 qe:0.06	comp_8 m:9.77 qe:0.06 comp_6 m:8.62 qe:0.08 comp_9 m:7.87 qe:0.07	comp_8 m:9.76 qe:0.04 comp_6 m:8.61 qe:0.06 comp_9 m:7.85 qe:0.05

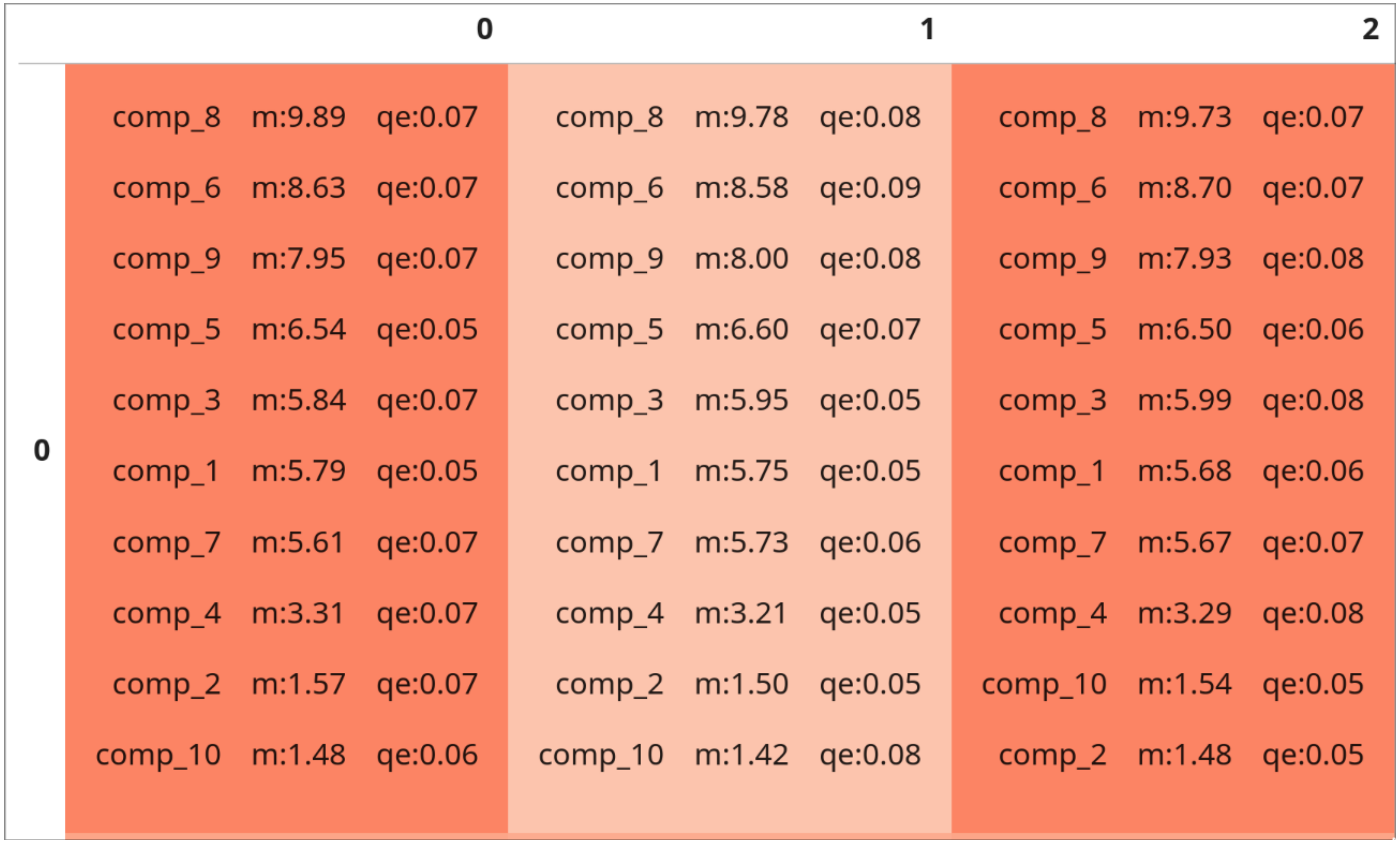


```
comp_5   m:6.59   qe:0.06      comp_5   m:6.57   qe:0.05      comp_5   m:6.70   qe:0.06

comp_3   m:5.88   qe:0.07      comp_3   m:5.98   qe:0.08      comp_3   m:5.93   qe:0.09
```

10clusters LabelSOM with 5 labels

Of course, our LabelSOM implementation can only print as many labels as there are features in the input data. Each input vector in the 10clusters dataset has 10 features (`comp_1` through `comp_10`). As a result, no more than 10 labels will be printed, even if `number_of_labels_to_generate` is set to 12 as in the following screenshot.



	0	1	2
0	comp_8 m:9.89 qe:0.07	comp_8 m:9.78 qe:0.08	comp_8 m:9.73 qe:0.07
	comp_6 m:8.63 qe:0.07	comp_6 m:8.58 qe:0.09	comp_6 m:8.70 qe:0.07
	comp_9 m:7.95 qe:0.07	comp_9 m:8.00 qe:0.08	comp_9 m:7.93 qe:0.08
	comp_5 m:6.54 qe:0.05	comp_5 m:6.60 qe:0.07	comp_5 m:6.50 qe:0.06
	comp_3 m:5.84 qe:0.07	comp_3 m:5.95 qe:0.05	comp_3 m:5.99 qe:0.08
	comp_1 m:5.79 qe:0.05	comp_1 m:5.75 qe:0.05	comp_1 m:5.68 qe:0.06
	comp_7 m:5.61 qe:0.07	comp_7 m:5.73 qe:0.06	comp_7 m:5.67 qe:0.07
	comp_4 m:3.31 qe:0.07	comp_4 m:3.21 qe:0.05	comp_4 m:3.29 qe:0.08
	comp_2 m:1.57 qe:0.07	comp_2 m:1.50 qe:0.05	comp_10 m:1.54 qe:0.05
	comp_10 m:1.48 qe:0.06	comp_10 m:1.42 qe:0.08	comp_2 m:1.48 qe:0.05

10clusters LabelSOM with 'number_of_labels_to_generate set to 12

Ignore labels with mean of zero

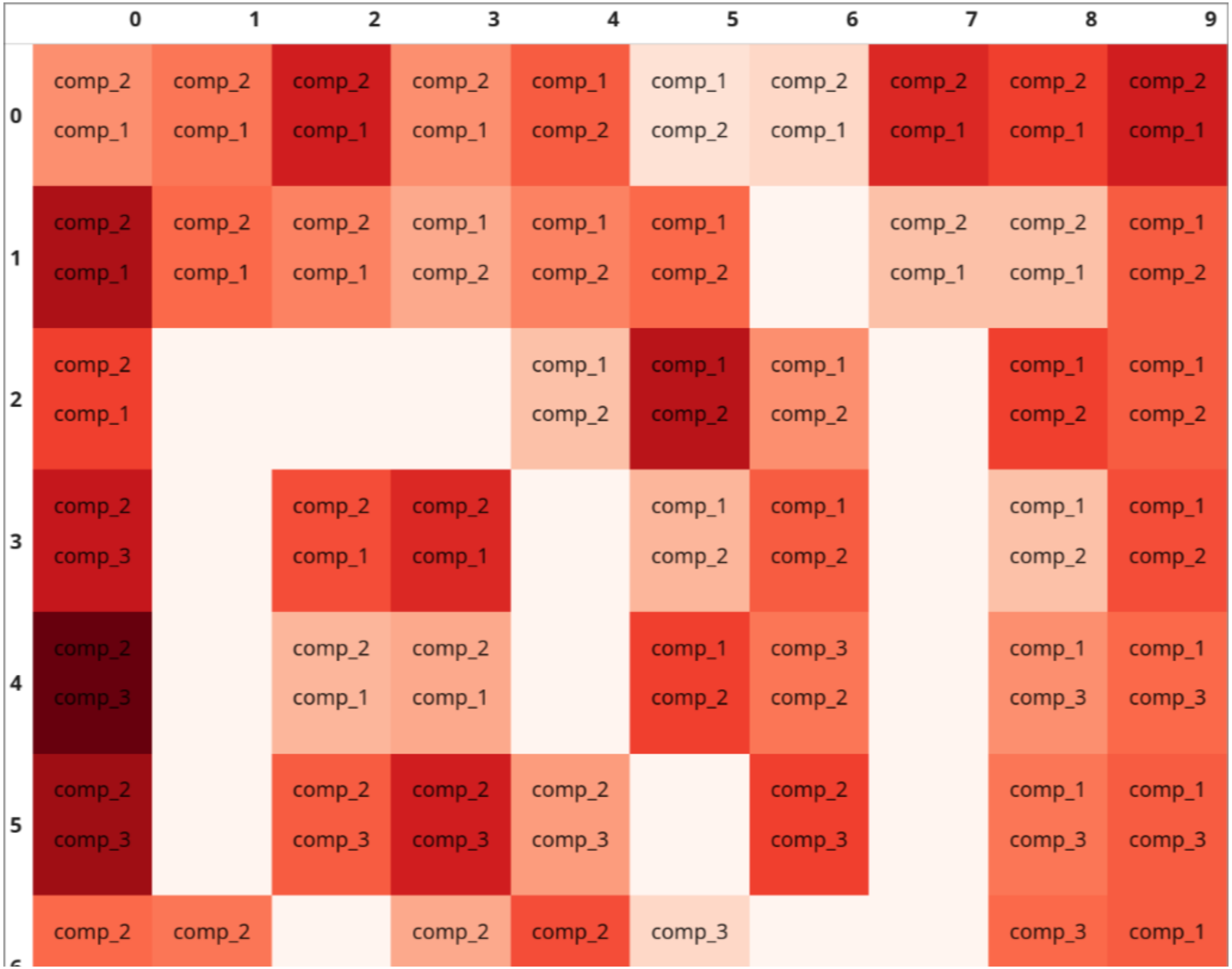
Although LabelSOM prioritizes features with a high mean value across all mapped input vectors, it can include labels with a mean of 0, if `number_of_labels_to_generate` is set to a high value. This behavior may not be desired, as the focus of LabelSOM is to identify features representative of the unit, whereas a mean value of 0 indicates a feature with no contribution to the unit's weight vector.

By setting the parameter `ignore_labels_with_zero` in `generate_label_matrix_and_hit_histogram` to `True` , features with a mean of 0 can be excluded. The same functionality is also present in the Java SOM toolbox.

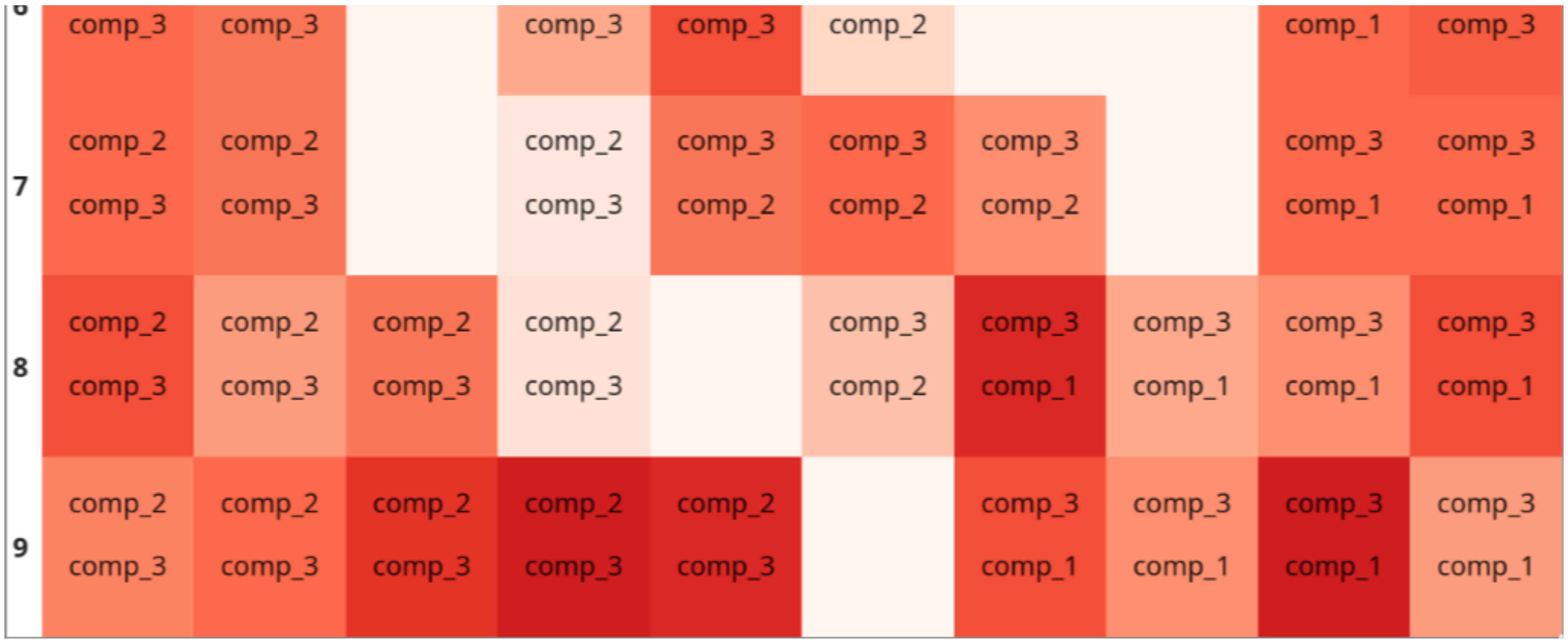
Hide mean or QE values

With a large number of labels and units, the visualization of our LabelSOM matrices can become cluttered. Setting the booleans `include_mean` and/or `include_quantization_error` to `False` in `pretty_print_label_matrix` can remedy this to an extent by reducing the amount of information displayed.

As their name implies, these parameters are used to display or hide the mean value and quantization error in the rendered table. Thus, it is possible to show only the title for each label in the LableSOM matrix, as displayed in the image below.



	0	1	2	3	4	5	6	7	8	9
0	comp_2 comp_1	comp_2 comp_1	comp_2 comp_1	comp_2 comp_1	comp_1 comp_2	comp_1 comp_2	comp_2 comp_1	comp_2 comp_1	comp_2 comp_1	comp_2 comp_1
1	comp_2 comp_1	comp_2 comp_1	comp_2 comp_1	comp_1 comp_2	comp_1 comp_2	comp_1 comp_2		comp_2 comp_1	comp_2 comp_1	comp_1 comp_2
2	comp_2 comp_1				comp_1 comp_2	comp_1 comp_2	comp_1 comp_2		comp_1 comp_2	comp_1 comp_2
3	comp_2 comp_3		comp_2 comp_1	comp_2 comp_1		comp_1 comp_2	comp_1 comp_2		comp_1 comp_2	comp_1 comp_2
4	comp_2 comp_3		comp_2 comp_1	comp_2 comp_1		comp_1 comp_2	comp_3 comp_2		comp_1 comp_3	comp_1 comp_3
5	comp_2 comp_3		comp_2 comp_3	comp_2 comp_3	comp_2 comp_3		comp_2 comp_3		comp_1 comp_3	comp_1 comp_3
6	comp_2	comp_2		comp_2	comp_2	comp_3			comp_3	comp_1



chainlink LabelSOM without mean or quantization error

5. Comparison with Java SOM toolbox

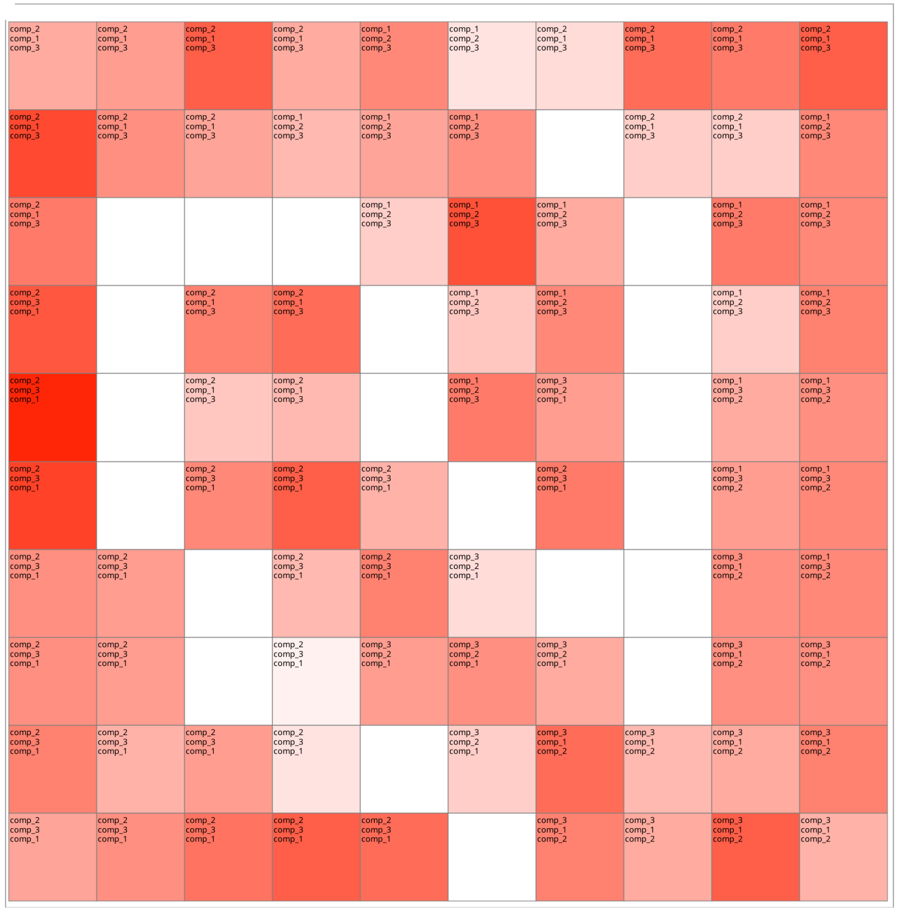
To validate our implementation of the LabelSOM algorithm, we compared its visualizations with those of the Java SOM toolbox. We found the results to be consistent between both programs: The Java SOM toolbox and our Python module assign the same labels to each SOM unit when given the same input data.

Comparison images with visualizations of both the chainlink and 10clusters datasets and corresponding 10x10 and 100x60 SOMs are shown below. Since the 100x60 SOM is too large to display here, we only render a zoomed-in view of the upper left corner for both the Python and Java output in this notebook. However, full color-coded output can be found in the images/ folder for the Java toolbox and the generated_visualizations/ folder for our Python implementation:

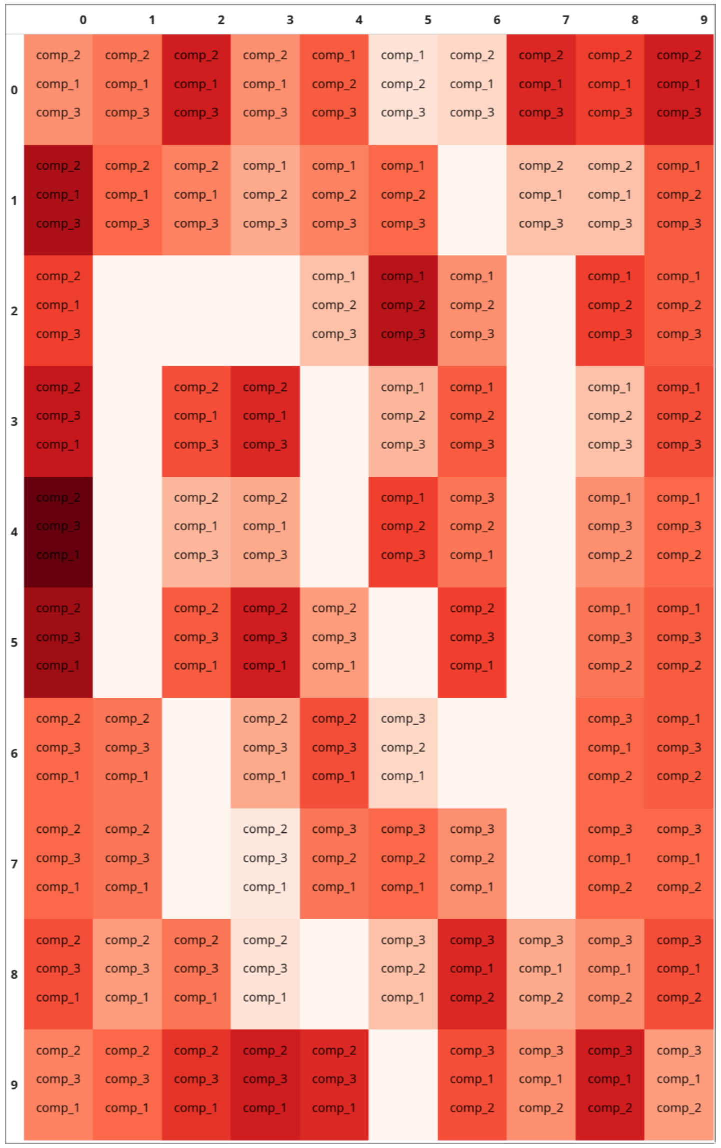
- Python 100x60 SOM for chainlink dataset
- Python 100x60 SOM for 10clusters dataset
- Java SOM toolbox 100x60 SOM for chainlink dataset

- Java SOM toolbox 100x60 SOM for 10clusters dataset

Chainlink dataset 10x10 SOM

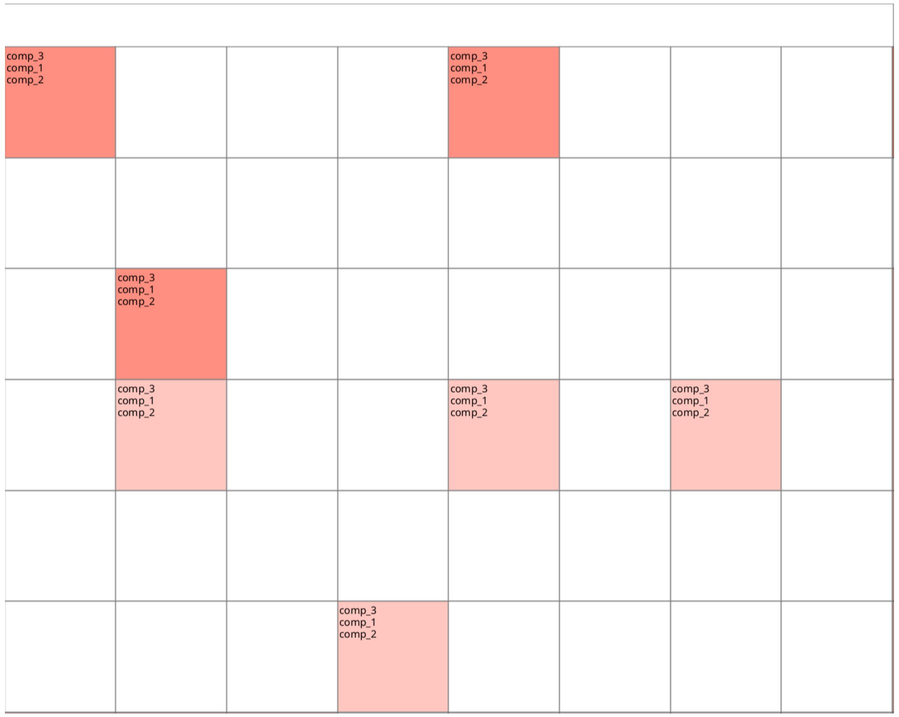


Java SOM toolbox implementation

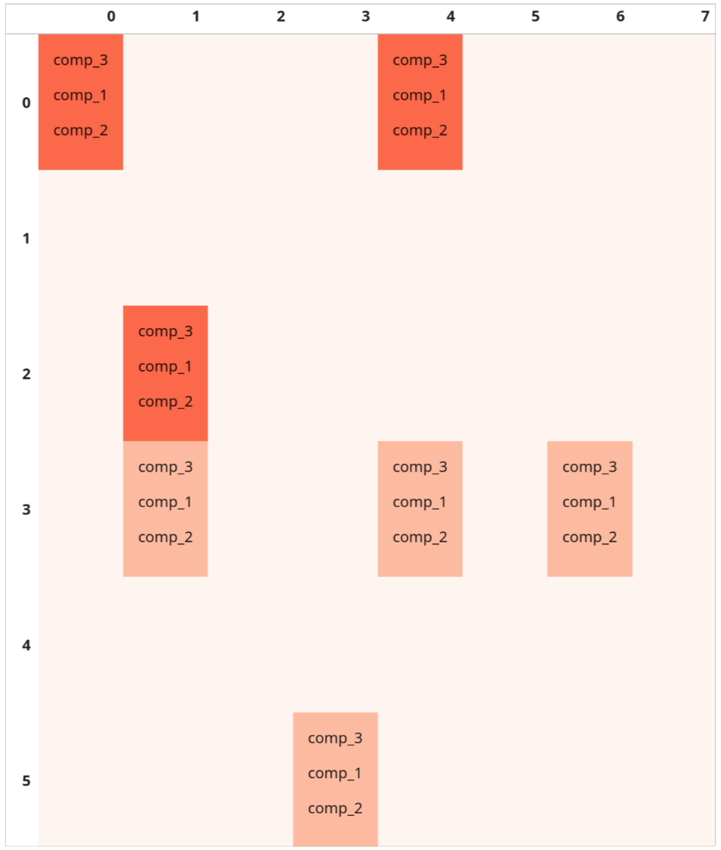


Python implementation

Chainlink dataset 100x60 SOM (zoomed on the upper left corner of the map)

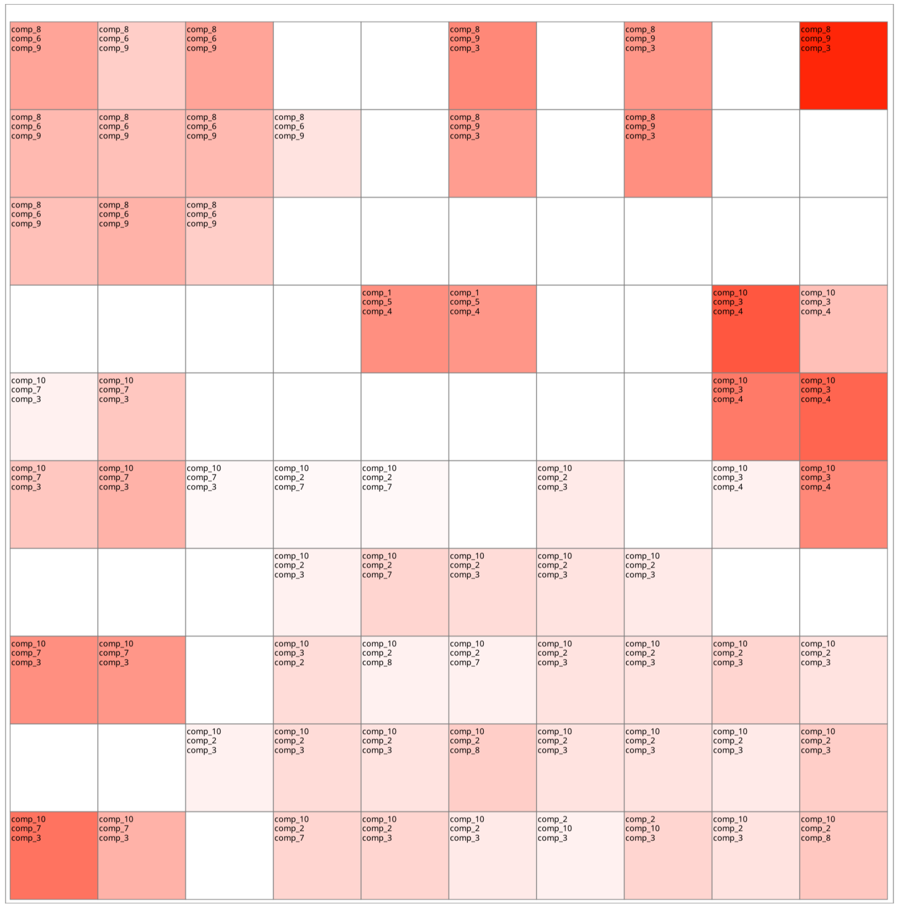


Java SOM toolbox implementation

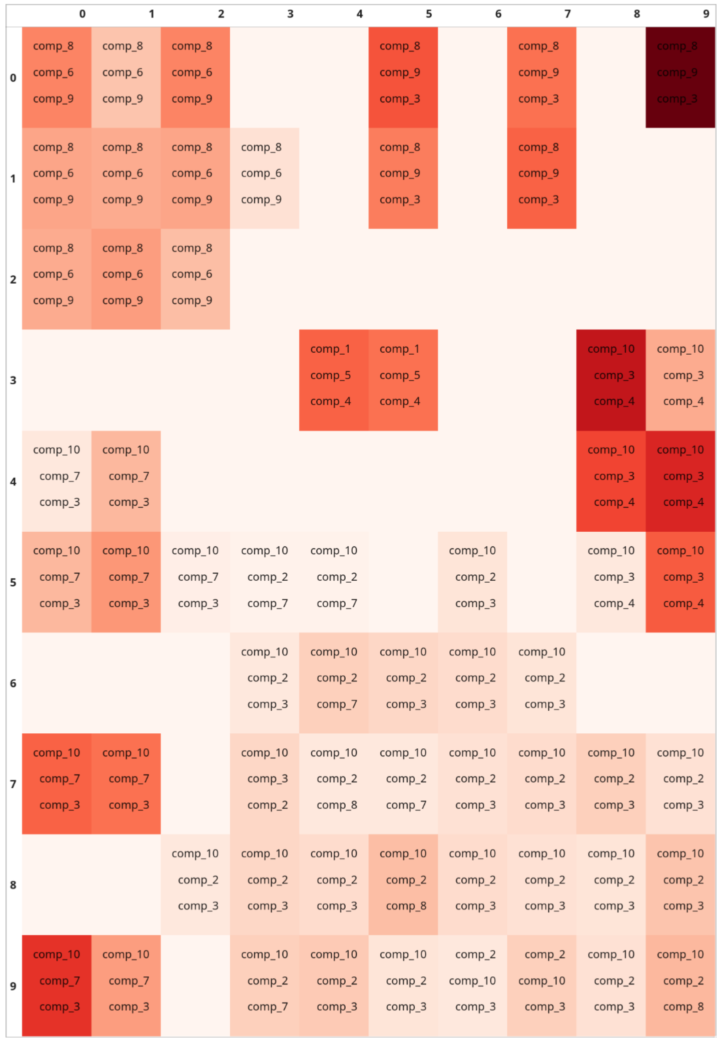


Python implementation

10clusters dataset 10x10 SOM

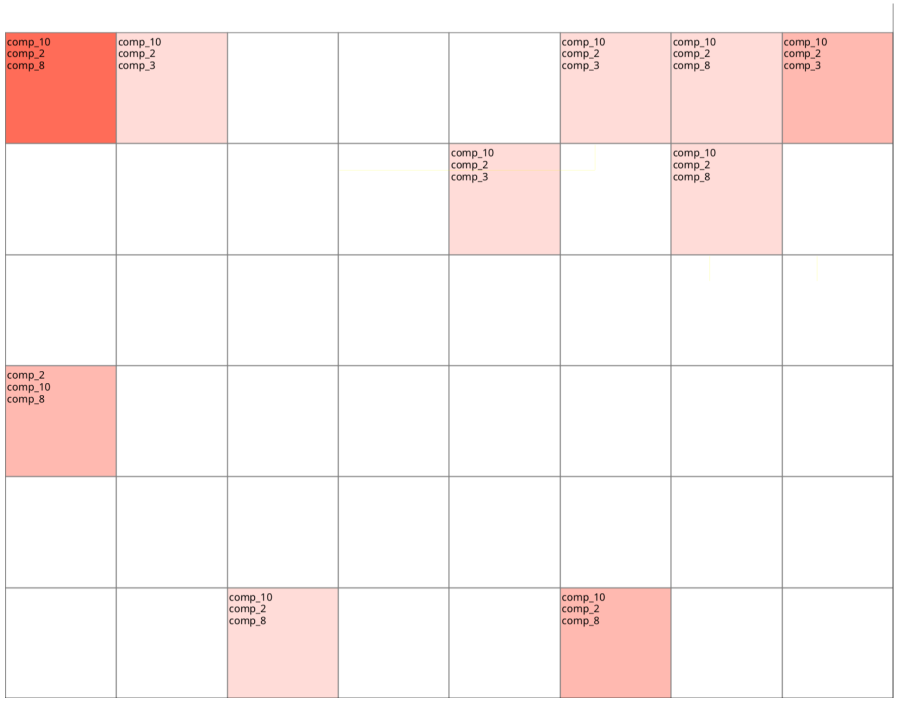


Java SOM toolbox implementation

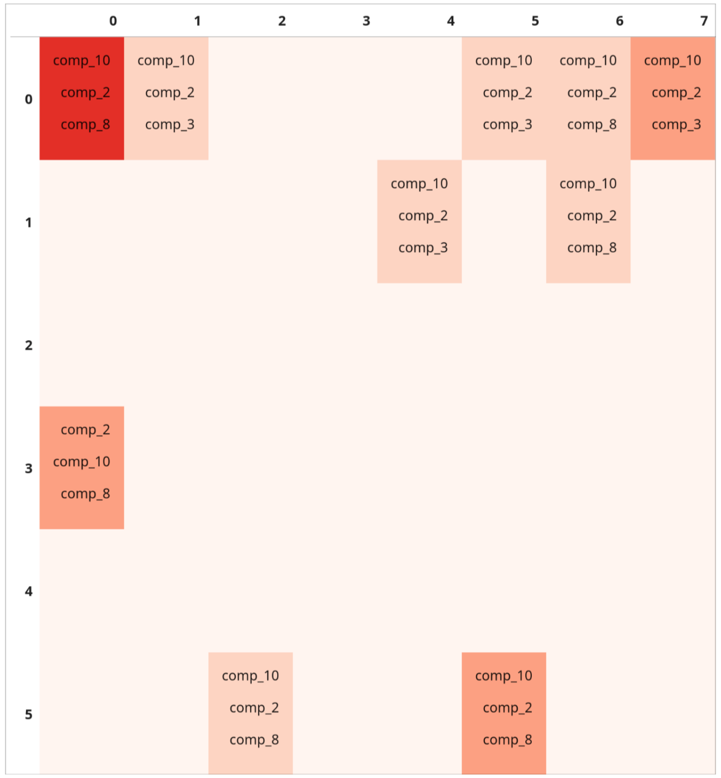


Python implementation

10clusters dataset 100x60 SOM (zoomed on the upper left corner of the map)



Java SOM toolbox implementation



Python implementation

Appendix

Dumped `labelsom` visualizations for all our trained SOMs can be found in `./generated_visualizations`. All dumped visualizations show the mean and quantization error for every lable. For the `chainlink` SOMs we generated 3 labels per unit and for the `10clusters` SOMs we generate 5 labels per unit.

- chainlink 10x10 LabelSOM
- chainlink 100x60 LabelSOM
- 10 clusters 10x10 LabelSOM
- 10 clusters 100x60 LabelSOM

In [11]:
```
# Generate visualizations for all trained SOMs:
```

```
# chainlink 10x10
# Already dumped in cell above
```

In [12]:
```
# chainlink 100x60
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/chainlink/100x60"))
label_matrix, hit_histogram = labelsom.generate_label_matrix_and_hit_histogram(
    m=m, n=n, weights=weights, input_data=input_data, attribute_names=attribute_names,
    number_of_labels_to_generate=3,
    ignore_labels_with_zero=False,
)
labelsom.write_labelsom_to_file(
    label_matrix,
    hit_histogram,
    include_mean=True,
    include_quantization_error=True,
    directory_to_write_file_to=Path("./generated_visualizations"),
    file_name="chainlink_100x60",
)
```

Out[12]:
```
PosixPath('generated_visualizations/chainlink_100x60.html')
```

In [13]:
```
# 10 Clusters 10x10
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/10clusters/10x10"))
label_matrix, hit_histogram = labelsom.generate_label_matrix_and_hit_histogram(
    m=m, n=n, weights=weights, input_data=input_data, attribute_names=attribute_names,
    number_of_labels_to_generate=5,
    ignore_labels_with_zero=False,
)
labelsom.write_labelsom_to_file(
    label_matrix,
    hit_histogram,
    include_mean=True,
    include_quantization_error=True,
    directory_to_write_file_to=Path("./generated_visualizations"),
    file_name="10clusters_10x10",
)
```

Out[13]:
```
PosixPath('generated_visualizations/10clusters_10x10.html')
```

In [14]:
```
# 10 Clusters 100x60
m, n, attribute_names, weights, input_data = read_in_trained_SOM(Path("./trainedData/10clusters/100x60"))
```

```python
label_matrix, hit_histogram = labelsom.generate_label_matrix_and_hit_histogram(
    m=m, n=n, weights=weights, input_data=input_data, attribute_names=attribute_names,
    number_of_labels_to_generate=5,
    ignore_labels_with_zero=False,
)
labelsom.write_labelsom_to_file(
    label_matrix,
    hit_histogram,
    include_mean=True,
    include_quantization_error=True,
    directory_to_write_file_to=Path("./generated_visualizations"),
    file_name="10clusters_100x60",
)
```

```
Out[14]: PosixPath('generated_visualizations/10clusters_100x60.html')
```