# iris10x10

January 29, 2025

[1]:
```
import copy
```

## 1 Iris dataset - 10x10 SOM

[2]:
```
from SOMToolBox_Parse import SOMToolBox_Parse

dataset = 'iris' #iris, chainlink, 10clusters, BostonHousing
inputdata = SOMToolBox_Parse('datasets/'+dataset+'/'+dataset+'.vec').
 ↪read_weight_file()
components = SOMToolBox_Parse('datasets/'+dataset+'/'+dataset+'.tv').
 ↪read_weight_file()
classinfo = SOMToolBox_Parse('datasets/'+dataset+'/'+dataset+'.cls').
 ↪read_weight_file()
```

### 1.1 Train SOM

[56]:
```
from pysomvis import PySOMVis
from minisom import MiniSom

n=10
m=10
dim=4
som = MiniSom(m, n, dim, random_seed=1)
som.train(inputdata['arr'], 10000)
```

#### 1.1.1 Save machine-actionable description of the model using FAIR4ML schema

[4]:
```
import json
import inspect
from jsonschema import validate, ValidationError

# Define the FAIR4ML schema (simplified example)
fair4ml_schema = {
    "type": "object",
    "properties": {
        "model": {"type": "string"},
        "parameters": {"type": "object"},
```

```
        "training_data": {"type": "string"},
        "iterations": {"type": "integer"}
    },
    "required": ["model", "parameters", "training_data", "iterations"]
}

# Function to collect metadata
def collect_metadata(model, training_data, iterations):
    parameters = inspect.signature(model.__init__).parameters
    parameters_dict = {k: str(v) for k, v in parameters.items()}  # Convert to␣
 ↪serializable format
    metadata = {
        "model": model.__class__.__name__,
        "parameters": parameters_dict,
        "training_data": training_data,
        "iterations": iterations
    }
    return metadata

metadata = collect_metadata(som, 'idata["arr"]', 10000)

# Validate metadata against the FAIR4ML schema
try:
    validate(instance=metadata, schema=fair4ml_schema)
    print("Metadata is valid.")
except ValidationError as e:
    print("Metadata validation error:", e)

# Export metadata to JSON-LD
with open('som_iris10x10_metadata.jsonld', 'w') as f:
    json.dump(metadata, f, indent=4)
print("Metadata exported.jsonld")
```

```
Metadata is valid.
Metadata exported.jsonld
```

### 1.1.2 Interactive PySOMVis visualization

Used to experiment different SOM visualizations, parameters, and color schemes.

[57]:
```
pysomviz = PySOMVis(weights=som._weights.reshape(100,4), m=m, n=n,
                    dimension=dim, input_data=inputdata['arr'],␣
 ↪classes_names=classinfo['classes_names'],
                   classes=classinfo['arr'][:,1],␣
 ↪component_names=components['arr'][:,1])
pysomviz._mainview
```

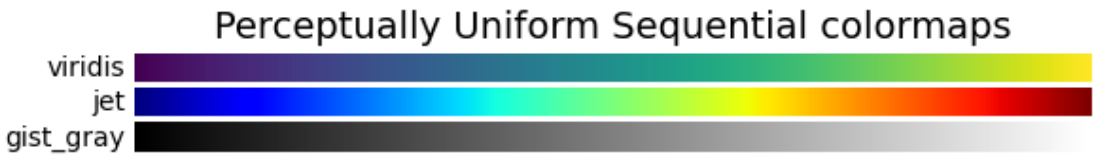
[57]:
```
Column
    [0] Column
        [0] Column
            [0] Row(margin=5, width=700)
                [0] StaticText(value='<b></b>')
                [1] Select(options=OrderedDict([('Component P…]), value=0)
                [2] Select(options=OrderedDict([('PiYG', …]), value='jet')
                [3] Button(name=' ')
                [4] Button(name=' ')
                [5] Button(name=' ')
                [6] Button(name=' ')
                [7] Checkbox(name='interpolation')
        [1] Row
            [0] Column
                [0] HoloViews(DynamicMap)
            [1] Row
                [0] Column
                    [0] Column(margin=5, name='Component Planes', width=300)
                        [0] StaticText(value='<b>Component Planes</b>')
                        [1] IntSlider(end=3, name='sep_length')
                    [1] Str(str)
    [1] Column
        [0] Row()
```

[9]:
```
for vis in pysomviz._visualizations:
    print(vis)
```

```
<visualizations.complane.ComponentPlane object at 0x7791bafecdf0>
<visualizations.hithistogram.HitHist object at 0x7791bafece50>
<visualizations.umatrix.UMatrix object at 0x7791bafece80>
<visualizations.dmatrix.DMatrix object at 0x7791bafeceb0>
<visualizations.upmatrix.UStar_PMatrix object at 0x7791baf80040>
<visualizations.sdh.SDH object at 0x7791baf801c0>
<visualizations.piechart.PieChart object at 0x7791baf802b0>
<visualizations.neighbourhood_graph.NeighbourhoodGraph object at 0x7791baf80610>
<visualizations.chessboard.Chessboard object at 0x7791baf807f0>
<visualizations.clustering.Clustering object at 0x7791baf80970>
<visualizations.metromap.MetroMap object at 0x7791baf80ac0>
<visualizations.qerror.QError object at 0x7791baf80b50>
<visualizations.somstreamvis.SOMStreamVis object at 0x7791baf80cd0>
<visualizations.sky_metaphor.SkyMetaphor object at 0x7791bafa3160>
<visualizations.topographic_error.TopographicError object at 0x7791bafa3400>
<visualizations.intrinsic_distance.IntrinsicDistance object at 0x7791bafa3610>
<visualizations.activityhist.ActivityHist object at 0x7791bafa3670>
<visualizations.minimumSpanningTree.MinimumSpanningTree object at
0x7791bafa37f0>
<visualizations.cluster_connection.ClusterConnection object at 0x7791bafa38b0>
<mnemonics.mnemonicSOM.MnemonicSOM object at 0x7791bafa3ca0>
```

### 1.2 Visualization of used colormaps

[10]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
cmaps = {}

gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))


def plot_color_gradients(category, cmap_list):
    nrows = len(cmap_list)
    figh = 0.35 + 0.15 + (nrows + (nrows - 1) * 0.1) * 0.22
    fig, axs = plt.subplots(nrows=nrows + 1, figsize=(6.4, figh))
    fig.subplots_adjust(top=1 - 0.35 / figh, bottom=0.15 / figh,
                        left=0.2, right=0.99)
    axs[0].set_title(f'{category} colormaps', fontsize=14)

    for ax, name in zip(axs, cmap_list):
        ax.imshow(gradient, aspect='auto', cmap=mpl.colormaps[name])
        ax.text(-0.01, 0.5, name, va='center', ha='right', fontsize=10,
                transform=ax.transAxes)
    for ax in axs:
        ax.set_axis_off()

    cmaps[category] = cmap_list

plot_color_gradients('Perceptually Uniform Sequential', ['viridis', 'jet',␣
 ↪'gist_gray'])
```




[11]:
```
import panel as pn
import holoviews as hv
from holoviews import opts
from skimage.transform import resize
hv.extension('bokeh')
```

[12]:
```
def calculate(visualization_index=0, cmap="jet", title="Visualization",␣
↪**kwargs):
    print(f"Calculating for {pysomviz._visualizations[visualization_index].
↪__class__.__name__}")
    pysomviz._visualizations[visualization_index]._calculate(**kwargs)
    return hv.Image(pysomviz._pipe.data).relabel(title).opts(width=400,␣
↪height=400).opts(cmap=cmap)
```

### 1.3 Component Planes

The Component Planes visualization displays how an individual feature is distributed across the SOM. Each SOM unit is represented as a cell, with color intensity indicating the magnitude of the selected feature value.

[13]:
```
calculate(0, "jet", title=f"Component Planes with comp petal length",␣
↪component=2)
```

```
Calculating for ComponentPlane
```

[13]:
```
:Image   [x,y]   (z)
```

The above Component Planes visualization allows to examine the variation of **petal length** across the SOM, clearly displaying the role of this feature in distinguing three clusters, each likely corresponding to "Setosa", "Versicolor", and "Virginica". One cluster is represented in the upper left corner, another in the lower right corner, and yet another in between the other two. This can likely be interpreted as one of the clusters representing data points with higher values of "petal lenght", another one representing data points with lower values of "petal lenght", and the "in between" cluster representing data points with values of "petal lenght that are neither the heighest nor the lowest in the dataset.

[14]:
```
calculate(0, "jet", title=f"Component Planes with comp petal width",␣
↪component=3)
```

```
Calculating for ComponentPlane
```

[14]:
```
:Image   [x,y]   (z)
```

The above Component Planes visualization allows to examine the variation of **petal width** across the SOM, clearly displaying the role of this feature in distinguing the same three clusters as petal length. Even tough being similar to the previous visualization, this shows the upper left cluster spreading to the middle of the visualization, mixing in with the "in between" cluster, and almost reacging the cluster in the lower right corner.

[15]:
```
calculate(0, "jet", title=f"Component Planes with comp sepal width",␣
↪component=1)
```

```
Calculating for ComponentPlane
```

[15]:
```
:Image   [x,y]   (z)
```

The above Component Planes visualization allows to examine the variation of **sepal width** across the SOM, once more displaying the role of the feature in distinguing clusters. But this feature, seems to distinguish the data in a completly different manner, there seems to be a small (red) cluster, another (blue) cluster represented mainly in the lower left corner, but also in the upper right corner and even one cell in the lower right corner, and the rest of the are represented in yellow-ish.

### 1.4 Hit- Histogram

The Hit Histogram shows the number of data points mapped to each SOM unit. Each unit is represented as a cell, with color intensity indicating the number of hits.

[16]:
```
calculate(1, "viridis", title="Hit Histogram")
```

```
Calculating for HitHist
```

[16]: :Image [x,y] (z)

## 2 TODO: interpret HitHist

### 2.1 U-Matrix

The Unified Distance Matrix (U-Matrix) allows one to visualize the distances between SOM units. Each unit is represented as a cell, with color intensity indicating the distance to its neighbors. High distances (yellow regions) suggest boundaries between clusters, while low distances (blue regions) indicate cluster homogeneity.

U-Matrix helps identify cluster structures and their boundaries within the SOM: dense regions of low distances can be interpreted as data clusters, while areas with high distances usually suggest separation between clusters.

[17]:
```
calculate(2, title="Calculating for UMatrix")
```

```
Calculating for UMatrix
```

[17]: :Image [x,y] (z)

The U-Matrix reveals two main well seperated clusters. The first cluster - most likely corresponding to “Setosa” - is well-separated with low intra-cluster distances, represented in the U-Matrix above in the lower right corner. The cluster “on the other side” of the boundry - likely corresponding to “Versicolor” and “Virginica” - represent most of the area in the visualization, probably due to the fact that there are twice as much data points that are either “Versicolor” or “Virginica” than there are “Setosa”. An explanation on why “Versicolor” and “Virginica” overlap and are represented in a single cluster is most probably due to the known difficulty in linearly separate these two classes.

### 2.2 D-Matrix

The Density Matrix (D-Matrix) visualizes the density of data points mapped to each SOM unit. Each unit is represented as a cell, with color intensity indicating the number of data points assigned to that unit. High-density regions are shown with more intense colors (blue), while low-density

regions are lighter (yellor/red). Dense regions indicate areas of the SOM that represent many similar data points, while sparse regions may reflect data outliers or underpopulated clusters.

[18]:
```
calculate(3, title = "D-Matrix")
```

```
Calculating for DMatrix
```

[18]:
```
:Image   [x,y]   (z)
```

The above D-Matrix visualization highlights two key regions of density, similarly to the U-Matrix. Again, the region represented in the bottom right cornern, likely corresponding to "Setosa," appears compact and highly concentrated with most data points mapped to a few closely packed units. This is reflective of the well-separated nature of "Setosa".

In contrast, the remaining two species, "Versicolor" and "Virginica", show a broader, more dispersed distribution across multiple units represented in the upper part of the D-Matrix. Once more, vast region suggests their known difficulty of separation. However, this D-Matrix offers some insight that was more difficult to notice in the U-Matrix: the yellow-ish units in the upper part of the visualization, seem to almost stretch to the sharp border that separates "Setosa" from the rest, this could be a strong suggestion of possible two clusters, each corresponding to "Versicolor" and "Virginica". And, if it wasn't for the overlap between these two categories, possibly the narrow blue gap stretching between the bottom left corner and the upper would instead be "closed", and each of those corners would represent a cluster, one corresponding to "Versicolor" and the other to "Virginica".

### 2.3 P-Matrix and U*-Matrix

[19]:
```
calculate(4, title = "...")
```

```
Calculating for UStar_PMatrix
```

[19]:
```
:Image   [x,y]   (z)
```

### 2.4 Smoothed Data Histograms

[20]:
```
calculate(5, title = "Smoothed Data Histograms", approach=0, factor=5)
```

```
Calculating for SDH
```

[20]:
```
:Image   [x,y]   (z)
```

[21]:
```
calculate(5, title = "Weighted Smoothed Data Histograms", approach=1, factor=5)
```

```
Calculating for SDH
```

[21]:
```
:Image   [x,y]   (z)
```

[22]:
```
calculate(5, title = "Weighted (norm.) Smoothed Data Histograms", approach=2,␣
↪factor=5)
```

```
Calculating for SDH
```

[22]: :Image [x,y] (z)

### 2.5 Pie Chart - !!! NOT WORKING?

!!! doesn't work - technically not a visualization per se, but a extra to add on top of other vis

[23]:
```
calculate(6, title = "Pie Chart")
pysomviz._pdmap[0]
```

```
Calculating for PieChart
```

[23]: Bokeh(Figure)

### 2.6 Neighbourhood Graph - !!! NOT WORKING?

!!! doesn't work - technically not a visualization per se, but a extra to add on top of other vis

[24]:
```
image = calculate(7, title = "Neighbourhood Graph", method=1)
image = copy.copy(image)
data = copy.deepcopy(pysomviz._pipe_paths.data)
pipe = hv.streams.Pipe(data=data)
dmap =  hv.DynamicMap(hv.Segments, streams=[pipe]).apply.opts(
    alpha='alpha',
    line_width=2,
    color="red"
)
image * dmap
```

```
Calculating for NeighbourhoodGraph
```

[24]:
```
:DynamicMap   []
   :Overlay
      .Image.Neighbourhood_Graph :Image   [x,y]   (z)
      .Segments.I                :Segments   [x0,y0,x1,y1]
```

### 2.7 Chess board

[25]:
```
calculate(8, title = "Chessboard")
```

```
Calculating for Chessboard
```

[25]: :Image [x,y] (z)

### 2.8 Clustering

[26]:
```
calculate(9, title = "Clustering with KMeans", method=0)
```

```
Calculating for Clustering
```

[26]: :Image [x,y] (z)

[27]:
```
# calculate(9, title = "Clustering", method=1)
```

### 2.9 Metro Map - !!! NOT WORKING?

when i run the bellow command, the interactive vis in the beggining of the notebook updates but the vis bellow shows a wrong vis?

[28]:
```
calculate(10, title = "Metro Map", calculating=True)
pysomviz._pdmap[0]
```

```
Calculating for MetroMap
```

[28]:
```
HoloViews(Overlay)
```

### 2.10 Quantization Error

[29]:
```
calculate(11, title = "Quantization Error", approach=0)
```

```
Calculating for QError
```

[29]:
```
:Image   [x,y]   (z)
```

[53]:
```
calculate(11, title = "Quantization Error", approach=1)
```

```
Calculating for QError
```

[53]:
```
:Image   [x,y]   (z)
```

### 2.11 SkyMetaphor - !!! NOT WORKING? + USE ANOTHER COLOR SCHEME

when i run the bellow command, the interactive vis in the beggining of the notebook updates but the vis bellow shows a wrong vis?

[58]:
```
image_sky = calculate(13, title = "SkyMetaphor", cmap="viridis")

points = copy.deepcopy(pysomviz._pipe_points.data)
point_map = hv.DynamicMap(hv.Points, streams=[hv.streams.Pipe(data=points)])
image_sky * point_map
```

```
Calculating for SkyMetaphor
```

[58]:
```
:DynamicMap   []
   :Overlay
      .Image.SkyMetaphor :Image   [x,y]   (z)
      .Points.I          :Points   [x,y]
```

### 2.12 Intrinsic Distance

[60]:
```
calculate(15, title = "Intrinsic Distance")
```

```
Calculating for IntrinsicDistance
```

[60]: :Image [x,y] (z)

### 2.13 Activity Histogram

[61]:
```
calculate(16, title = "Activity Histogram w/ input vec. 30", vec_idx=30)
```

```
Calculating for ActivityHist
```

[61]: :Image [x,y] (z)

[63]:
```
calculate(16, title = "Activity Histogram w/ input vec. 60", vec_idx=70)
```

```
Calculating for ActivityHist
```

[63]: :Image [x,y] (z)

[72]:
```
calculate(16, title = "Activity Histogram w/ input vec. 110", vec_idx=110)
```

```
Calculating for ActivityHist
```

[72]: :Image [x,y] (z)

### 2.14 Cluster Connection - !!! NOT WORKING?

what is 'paths' arg?

[77]:
```
pysomviz = PySOMVis(weights=som._weights.reshape(100,4), m=m, n=n,
                    dimension=dim, input_data=inputdata['arr'],␣
 ↪classes_names=classinfo['classes_names'],
                   classes=classinfo['arr'][:,1],␣
 ↪component_names=components['arr'][:,1])
pysomviz._mainview
```

[77]:
```
Column
    [0] Column
        [0] Column
            [0] Row(margin=5, width=700)
                [0] StaticText(value='<b></b>')
                [1] Select(options=OrderedDict([('Component P…]), value=0)
                [2] Select(options=OrderedDict([('PiYG', …]), value='jet')
                [3] Button(name=' ')
                [4] Button(name=' ')
                [5] Button(name=' ')
                [6] Button(name=' ')
```

```
                [7] Checkbox(name='interpolation')
            [1] Row
                [0] Column
                    [0] HoloViews(DynamicMap)
                [1] Row
                    [0] Column
                        [0] Column(margin=5, name='Component Planes', width=300)
                            [0] StaticText(value='<b>Component Planes</b>')
                            [1] IntSlider(end=3, name='sep_length')
                        [1] Str(str)
        [1] Column
            [0] Row()
```

[ ]: