HandsOn Assignments: SOM

This is the description of the "coding" option for assignment 3, i.e. for groups wishing to implement SOM visualizations in Python rather than perform extensive experiments using the SOM. It focuses on implementing one of a set of visualizations, providing a short report on how it was implemented, how it is to be used, as well as a few examples of visualizations obtained.

A) Jupyter notebook with selected SOM visualizations

- You will receive a Jupyter notebook (*coding_assignment.ipynb*, available at https://github.com/smnishko/PySOMVis) that contains the modules for SOM training and some basic visualizations (U-matrix, Hit histogram, SDH), to which you should add your own code. The notebook also contains code to read SOM files trained using the Java SOM toolbox so that you can compare the visualizations you create with the ones provided by the toolbox.

B) Implementation

1) ***Select and register* the (group of) visualizations** you picked for implementation with your group number in the TUWEL Wiki. (first come, first serve - do it early to be able to choose a visualization you also find interesting to work on)
2) **Implement the (set of) visualizations** according to the group you selected. You may, of course, refer to the source code of the Java SOM toolbox for inspiration on the coding, or consult other external sources. Also, use (solid) libraries wherever possible.
3) Provide a documentation of the implementation and explanations, ideally integrated as text directly in the Jupyter Notebook (to be exported as PDF for the final report, or, alternatively, as a separate report)
4) Implementations should be provided via a public repository (e.g. Github) and specify an Open Source license, preferably Apache (or MIT).

Visualizations to choose from are listed below (some containing several sub-visualizations):

a) Dendrogram-based visualizations: (as line overlay on coloring): connecting units based on a set of aggregate functions (using carefully selected (!) existing libraries for computing the aggregates), with parameters allowing to emphasize different edge width for tree structures so that the root and leave nodes become clearly visible (or for other forms of edge weights, such as distance, frequency, …) for (1) Minimum spanning tree (MST) over weight vectors (using an existing library for computing the MST); Hierarchical clustering of weight vectors using the different clustering algorithms provided i.e. (2) single-linkage, (3) complete-linkage, (4) average-linkage, and (5) WARD.

b) Vector Fields: Flow and Borderline visualizations as line overlay with a parameter to select the kernel size, supporting up to three groups of attributes (to be selected manually by the user by assigning each attribute to group 1, 2 or 3 – or none).

c) Visualization of the Topographic product and its components, i.e. (1) distortion in input space, (2) distortion in output space, (3) mean of the two as topographic product

d) LabelSOM: print the names of the n attributes per unit/cluster that show the lowest variance and/or the highest mean values within a specific unit or cluster, as text on the map units / clusters.

e) Visualizing attribute values per unit: create a number of plots that show – per unit – the values of (selected) attributes (feature vector components). Allow users to select which attributes to

display, as plotting 100s of them does not make much sense. Plot them – using existing libraries wherever possible - as

i) Bar charts
ii) Line charts (note: these obviously only make sense when the attributes show some form of sorting)
iii) Radar chart / Spider Web chart
iv) "Chernoff faces": using according libraries to create metaphor graphics to depict attribute values as overlays on units (or, as averages, for regions/clusters), allowing for some flexibility in the mapping between attributes and metaphor features.
v) BoxPlots (using the data instances mapped onto each unit to compute the distribution per (selected) attribute)
vi) Violin plots (using the data instances mapped onto each unit to compute the distribution per (selected) attribute)

f) SOM Comparison: visualizing "stable" and "outlier" shifts between either two maps (as arrows) or aggregated as coloring comparing one against n maps. For the comparison of two maps via arrows indicating shifts, provide a parameter that allows to specify a neighborhood radius on the second SOM for shifts to be still counted as "stable".

g) Mnemonic SOM: Reading a black/white silhouette image to determine the shape of the SOM; define max extension for units x and y and fit into silhouette; computing an (efficient) look-up table for shortest distance between two units for the training process, and mapping it into a regular SOM data structure to support the standard visualizations.

h) Aligned SOMs: Training a layer of n SOMs with differently weighted subsets of attributes (using the standard training functions implemented in SOM packages), storing/exporting them as individual or linearly arranged SOMs, i.e. 1) create two sub-sets of the feature vector by assigning attributes to group A and group B; 2) create n SOM layers initialized identically (i.e. the same random values) weighed by the respective layer weight vector (from 0 for group A attributes and 1 for group B to 0/a for groups A and B in n steps; 3) train aligned SOMs by randomly selecting an input vector and map layer, weigh the input vector with the according layer weight (0/1 to 1/0 for groups A and B in steps of n), training the respective SOM layer and the neighboring layers using the standard neighborhood function; 4) plot the SOM layers as individual maps next to each other. (cf. slide set and: Elias Pampalk. Aligned Self-Organizing Maps. Proc. of the Workshop on Self-Organizing Maps (WSOM'03), pp 185-190, Japan, 2003.)

i) Weighted Class Coloring: Enhance the class coloring visualizations (Chessboard/Floodfilling) by applying a weighting of the color intensity based on (selectable) SDH, (inverted) U-Matrix, P-Matrix and U*-Matrix, e.g. by transforming the Class coloring into an HSV colorspace and weighing the saturation channel according to the weighting selected by the user. If possible, try to allow for a parameter to modify the intensity of the saturation decrease with decreasing weight.

j) Create Demonstrator Notebooks: This assignment is slightly different in the light that you do not need to implement any new visualization, but that you should prepare an exhaustive demonstrator notebook of all the visualizations available in the PySOMVis library with according explanations / interpretations of the results obtained from different parameter settings for each of the visualizations. You will have, in total, two identical notebooks per data set, but with differing explanatory text (one each for the "normal sized" and one large SOM). Make sure you select interesting juxtapositions to show the effect of different parameter values (the same visualization may appear several times if juxtaposed with different other visualizations to obtain a clear interpretation and to show parallels / differences in information

provided). Choose meaningful combinations of colorings and line overlays, using suitable color palettes. You may, of course, cooperate across groups in creating the Jupyter Notebooks to create all visualizations, as only the parameters tested and the explanatory text will differ – specifically when it comes to cluster characteristics such as cardinality and density, or hierarchical structures of clusters. For the "classic benchmark data sets" discussed in class 3 datasets are combined per exercise as the class/cluster structure is well-known / pre-defined, requiring less analytical exploration than for the other benchmark data sets in Combo 3.

(1) Data Set Combo 1: Iris (10x10, 30x15) , Zoo (10x10, 60x40), and 10-Clusters (10x10, 100x60) (total of 6 notebooks, 2 per dataset) from http://www.ifs.tuwien.ac.at/dm/somtoolbox/datasets.html
(2) Data Set Combo 2: Chain Link (10x10, 60x40), Animals (5x5, 10x10), and Boston Housing (10x10, 100x60) (total of 6 notebooks, 2 per dataset) from http://www.ifs.tuwien.ac.at/dm/somtoolbox/datasets.html
(3) Data Set Cobo 3: (total of 4 notebooks, 2 per dataset):
   i. Room Occupancy: (60x40, 300x200) https://archive.ics.uci.edu/ml/datasets/Room+Occupancy+Estimation
   ii. Wine quality: (60x40, 200x150) https://archive.ics.uci.edu/ml/datasets/Wine+Quality

C) Evaluation Report

1) For evaluations of tasks a-h, pick the Chainlink Data Set and the 10-Clusters dataset from http://www.ifs.tuwien.ac.at/dm/somtoolbox/datasets.html
2) Train a 10x10 (small) and a 100x60 (large) SOM. Make sure that the SOMs are properly trained, i.e. that the structures to be expected in the SOM become clearly visible. (Note: if you run into performance issues, try running the code natively in Python, which is sometimes substantially faster than running it in a Jupyter notebook)
3) Show the visualizations, providing examples with different parameter settings and comparisons that allow a validation of the correctness of the implementation. Specifically, test a few extreme values for the parameter settings.
4) Where an identical visualization exists in the JÁVA SOM toolbox, read a SOM pre-trained with the JAVA SOM Toolbox (import functions are provided in the notebook) and compare your visualization with the one produced by the Java SOMToolbox (using either the pre-trained SOMs provided with the toolbox, or any that your colleagues who do the analytics option of the exercise share with you).
5) Describe, preferably directly within the Jupyter notebook (with options to select one of the two input files specified above and an arbitrary input file from local storage) and provide (export) as separate PDF report:
   - The implementation developed, explaining key parts of the code
   - The way to operate the code, i.e. adapting parameters, and discussing their effect on the visualization
   - Present the various evaluations performed under item 3) above, demonstrating the correctness of the implementation, and the plausibility of the information to be gained.
   - Compare the visualization with the identical visualizations (reading the same trained SOM files) using the SOM Java Toolbox

D) Feedback

1. (**optional**) Provide feedback on the exercise in general, and specifically whether having this option of a more implementation-oriented assignment makes sense, etc. (this section is,

obviously, optional and will not be considered for grading. You may also decide to provide that kind of feedback anonymously via the feedback mechanism in TISS – in any case we would appreciate learning about it to adjust the exercises for next year.)

HandsOn Assignments: SOM

Submission guidelines:

- **Upload ONE [zip/tgz/rar] file** to TUWEL that **contains all your files** (all scripts/programs you wrote and subsidiary information for repeating experiments) as well as the report document (PDF) export from the Jupyter Notebook ("print-to-PDF" or separate report) containing your names and student IDs as well as the link to the repository (e.g. Github)**.** You must follow this naming convention for the zip file and report document:
  - ***SOS2023_ExSOM_group<groupno>_<studID1>_<studID2>_<studID3>.[zip/tgz/rar]***
  - Example: A submission of group 99 with 3 students (ids: 019999, 029999, 039999) looks like this: SOS2023_ExSOM_group_coding_topic_g_0019999_0029999_0039999.[zip/tgz/rar]
  - Apply the same naming convention to the report (but obviously with pdf extension)
- **Put your names and your studentIDs in the report/notebook** (as author info).