

Homework1

SOM Clustering

SLFN Classification

Course:

Deep Learning

Supervisor:  
Dr. Mansoori

Atefe Rajabi

Spring 2024

Index

[Winner Takes All](#Winner)

Methodology

Implementation

Result

[On-Center Off-Surround](#OnCenter)

Methodology

Implementation

Result

[SLFN Classification](#SLFN)

Methodology

Implementation

Result

**Winner Takes All**

1. Methodology

Dataset Preparation:

The datasets provided contain cumulative counts of COVID-19 deaths, confirmed cases, and recoveries by country. Each dataset was processed to average the data for countries with multiple entries (representing different states or provinces). Latitude and longitude features were removed as they are not relevant to the clustering objective. The data was then standardized to ensure equal weighting during the clustering process.

SOM Implementation:

A linear SOM was implemented with four neurons, corresponding to four clusters intended to represent different levels of COVID-19 severity: Low, Medium, High, and Very High. The SOM was trained to adjust its weights to the input data, effectively grouping similar countries together based on their COVID-19 case profiles.

Clustering and Labeling:

After training, each country was assigned to one of the four clusters. The average case count within each cluster was calculated to determine the severity labels. These labels were assigned in ascending order based on the average, ensuring that clusters with lower averages received lower severity labels.

2. Implementation:

Libraries Import

- ‘numpy’: Used for numerical operations.

- ‘pandas’: Provides data structures and data analysis tools.

- ‘matplotlib.patches’, ‘matplotlib.pyplot’: Used for creating and customizing plots.

- ‘seaborn’: A visualization library based on matplotlib, providing a higher-level interface for drawing attractive statistical graphics.

- ‘geopandas’: Extends the datatypes used by pandas to allow spatial operations on geometric types, useful for plotting geographical data.

- ‘matplotlib.colors’: For handling color mappings in plots.

- ‘StandardScaler’ from ‘sklearn.preprocessing’: Standardizes features by removing the mean and scaling to unit variance, crucial for neural network inputs.

Class ‘LinearWTASOM’

- ‘\_\_init\_\_’: Initializes the SOM with randomly generated weights, and sets the number of neurons, input dimension, and learning rate.

- ‘find\_bmu’: Finds the Best Matching Unit (BMU) by computing the neuron with the least Euclidean distance to the input vector.

- ‘update\_weights’: Adjusts the weights of the BMU to be more like the input vector, based on the learning rate.

- ‘train’: Trains the SOM over a specified number of iterations, updating weights for each input vector.

- ‘predict’: After training, predicts the cluster (neuron index) of an input vector by finding its BMU.

Function ‘read\_dataset’

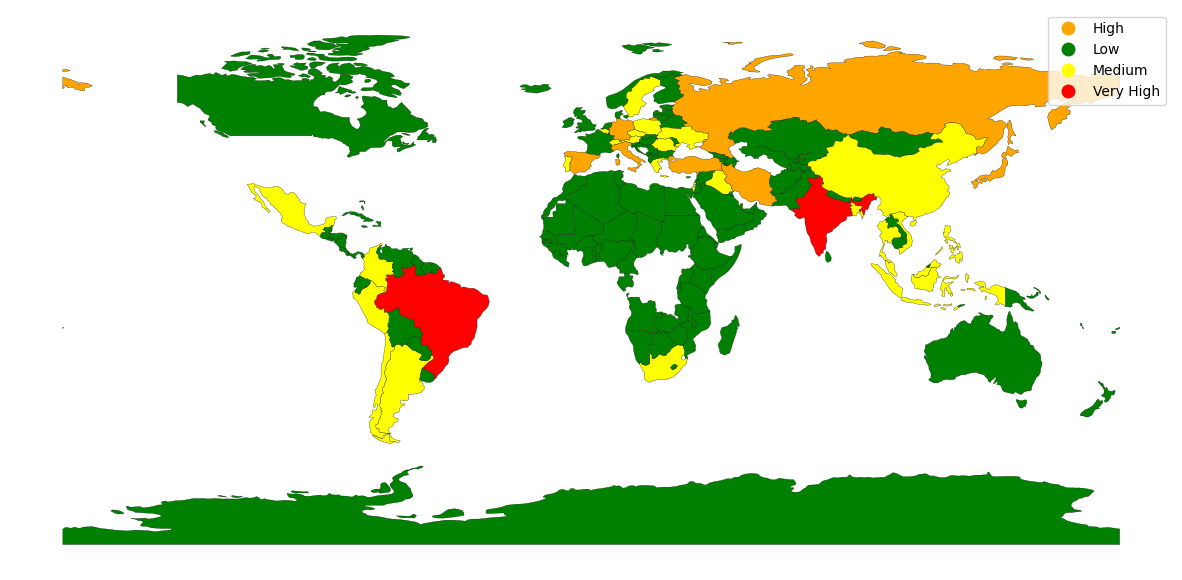
- Loads the dataset from a CSV file, averages data for countries with multiple entries, drops latitude and longitude, and standardizes the features using ‘StandardScaler’. Returns the scaled features, country names, and the processed DataFrame.

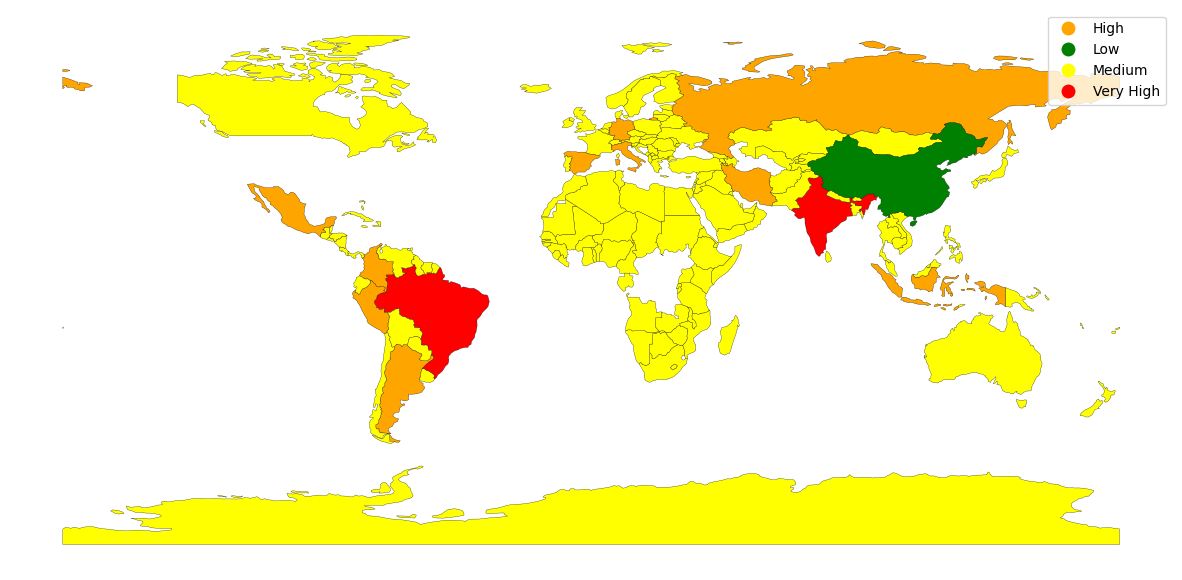
Function ‘cluster\_dataset’

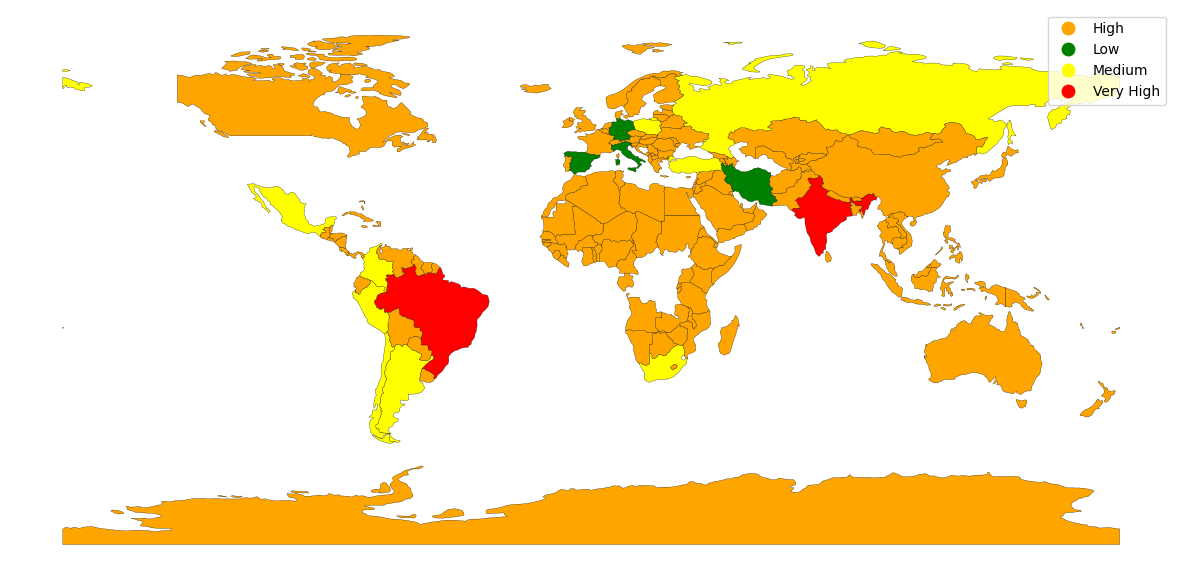
- Orchestrates the clustering process using the provided dataset. It involves reading and preprocessing the dataset, initializing and training the SOM, predicting cluster labels for each country, and assigning severity labels based on cluster averages. Finally, it merges the clustering results with a world map and plots it, coloring countries based on their assigned cluster for visual analysis.

4. Result:

1-time\_series\_covid19\_confirmed\_global.png



2-time\_series\_covid19\_deaths\_global.png

3-time\_series\_covid19\_recovered\_global.png

**On-Center Off-Surround**

1. Methodology

Data Preparation:

The dataset was standardized using ‘StandardScaler’ to ensure all features contribute equally to the analysis. The dataset was split into training and test sets with an 80-20 ratio.

SOM Training:

Two SOM grids (30x30 and 20x20) were trained on the dataset. Each grid represents a different level of granularity for the clustering.

Clustering Approaches:

1. Voting Approach: Labels for each neuron were determined by the majority vote of the training data points mapped to that neuron.

2. K-Means Approach: Applied the K-Means algorithm on the SOM's final weights to assign labels to each neuron, offering an alternative clustering perspective.

Evaluation Metrics:

The clustering quality was evaluated using the Davies-Bouldin Index (DBI), Normalized Mutual Information (NMI), and Adjusted Rand Index (ARI).

Implementation Details:

- The ‘MiniSom’ library was utilized to implement the SOMs. Training involved adjusting neuron weights over 10,000 iterations for the 30x30 grid and 1,000 iterations for the 20x20 grid.

- For the voting approach, a 3D array (‘winner\_map\_voting’) tracked the number of votes each neuron received for each label. The label with the majority votes was assigned to each neuron.

- The K-Means approach flattened the SOM weights into a 2D array for clustering. The optimal number of clusters was explored by varying the ‘n\_clusters’ parameter from 2 to 6 and evaluating the results using the defined metrics.

- Visualization of the neuron labels for both approaches provided insight into the clustering distribution across the SOM grid.

2. Implementation

Libraries:

MiniSom: SOM on-center off-surround network

Sklearn: metrics, KMeans, Standardization, One hot encoding

Mathplotlib numpy and pandas: for data manipulation and visualization Data Preparation and Preprocessing

- ‘remap\_labels(labels)’: This function takes a list of labels and remaps them according to a predefined dictionary. This is useful for adjusting label values to a desired range or format.

- Standardization and One-Hot Encoding: ‘StandardScaler’ is applied to the features to normalize them, ensuring each feature contributes equally to the analysis. ‘OneHotEncoder’ is used to encode categorical labels into a binary matrix, making them suitable for multi-class classification.

SOM Training and Clustering Approaches

- SOM Training: Utilizes the ‘MiniSom’ library to create and train a Self-Organizing Map on the training data. Two SOM configurations are trained: a 30x30 and a 20x20 grid, each with specific parameters for learning rate (‘sigma’) and neighborhood function.

- ‘voting\_approach(som, dim, X, y)’: Implements the voting mechanism to assign labels to SOM neurons based on the majority label of the data points mapped to each neuron. It plots the resulting neuron label map and returns a flattened array of neuron labels.

- ‘Kmeans\_approach(som, n\_clusters)’: Applies the K-Means clustering algorithm to the weights of the trained SOM to assign labels to neurons. The function plots the K-Means-based neuron label map and returns the neuron labels determined by K-Means.

Clustering Evaluation

- ‘test\_clusters(title, som, X, y, neuron\_labels)’: Evaluates the clustering performance on either the training or test dataset using the provided SOM and neuron labels from either the voting or K-Means approach. It calculates the Davies-Bouldin Index (DBI), Normalized Mutual Information (NMI), and Adjusted Rand Index (ARI) to assess clustering quality. The function prints these metrics and returns their values for further analysis.

Additional Functions

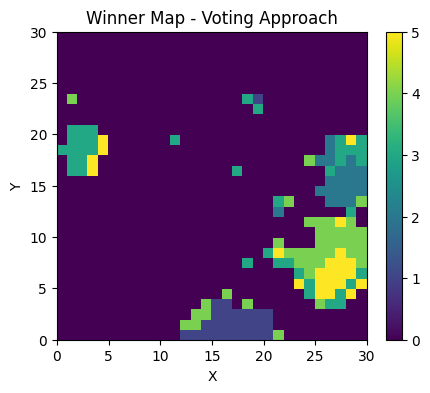
- Sigma Calculation: Calculates the initial ‘sigma’ value for the SOM training, based on the dimensions of the SOM grid. However, based on experience, initial Sigma = 1 has outperformed.

- Training and Evaluation: Trains SOM models with specified configurations, applies the voting and K-Means approaches to assign labels to SOM neurons, and evaluates the clustering results using the test dataset.

3. Result:

SOM 30x30:

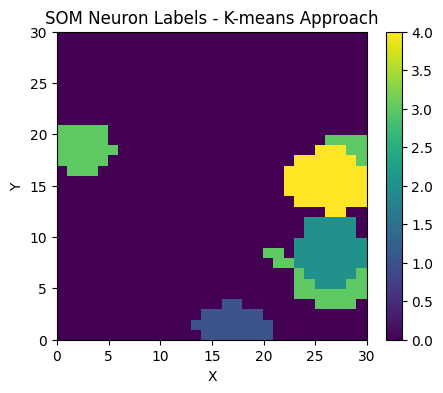
Train:

Voting Approach:

Davies-Bouldin Index: 0.84

Normalized Mutual Information: 0.75

adjusted rand score: 0.93

Kmeans Approach: (5 cluster)

Davies-Bouldin Index: 6.41

Normalized Mutual Information: 0.52

adjusted rand score: 0.82

Test:

Voting Approach:

Davies-Bouldin Index: 0.92

Normalized Mutual Information: 0.75

adjusted rand score: 0.93

Kmeans Approach:

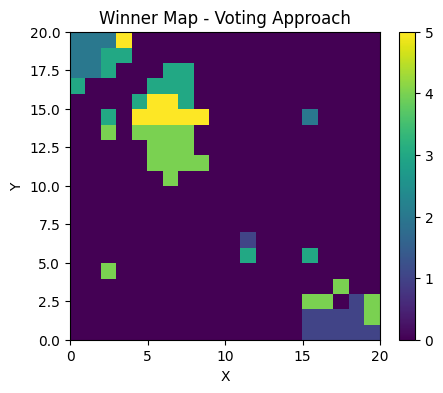
Davies-Bouldin Index: 6.51

Normalized Mutual Information: 0.53

adjusted rand score: 0.82

SOM 20x20:

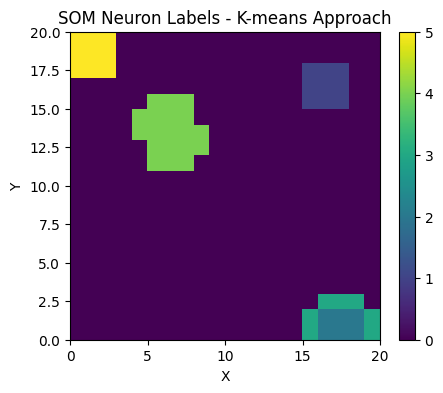
Train:

Voting Approach:

Davies-Bouldin Index: 8.45

Normalized Mutual Information: 0.19

adjusted rand score: 0.49

Kmeans Approach:

Davies-Bouldin Index: 2.11

Normalized Mutual Information: 0.53

adjusted rand score: 0.82

Test:

Voting Approach:

Davies-Bouldin Index: 23.48

Normalized Mutual Information: 0.22

adjusted rand score: 0.52

Kmeans Approach:

Davies-Bouldin Index: 5.15

Normalized Mutual Information: 0.53

adjusted rand score: 0.82

**SLFN Classification:**

1. Methodology

Data Preparation:

The dataset was loaded, and labels were remapped to a specified range. Features were standardized using ‘StandardScaler’ to normalize the data, facilitating effective training. The ‘OneHotEncoder’ was employed to encode categorical labels, making them suitable for the classification task.

SLFN Implementation:

A custom SLFN class was defined, including methods for the forward pass, prediction, accuracy calculation, and training. The network initializes weights and biases, utilizes a softmax activation function for output normalization, and employs cross-entropy loss for error measurement.

Training Process:

The network was trained on a subset of the data, using stochastic gradient descent to update weights and biases based on the calculated gradients. The training involved multiple epochs, with progress visualized in terms of loss and accuracy.

Evaluation:

Post-training, the SLFN's predictive capability was evaluated on both the training and test sets, and classification reports were generated to detail precision, recall, and F1-score for each class.

2. Implementation:

Data Loading and Preprocessing

- ‘remap\_labels(labels)’: This function takes an array of labels and remaps them according to a predefined dictionary. This is particularly useful for adjusting dataset labels to a specific range or format that is more suitable for the classification task.

- StandardScaler and OneHotEncoder: These preprocessing steps from ‘sklearn’ standardize the features and encode categorical labels, respectively. Standardization ensures that all features contribute equally to the model by giving them the same scale. OneHot encoding converts categorical labels into a format that can be provided to the model for training and prediction.

SLFN Implementation

- ‘SLFN’ Class: It contains methods for initializing the network, performing the forward pass, computing the softmax function, calculating cross-entropy loss, predicting labels, determining accuracy, and training the model.

- ‘\_\_init\_\_(self, input\_size, output\_size)’: Initializes the network with random weights and one biases based on the specified input and output sizes.

- ‘softmax(self, x)’: Applies the softmax function to the input array ‘x’, normalizing the outputs to a probability distribution over predicted output classes.

- ‘cross\_entropy\_loss(self, y\_true, y\_pred)’: Computes the cross-entropy loss between the true labels (‘y\_true’) and the predicted probabilities (‘y\_pred’), aiding in evaluating the performance of the network.

- ‘forward(self, x)’: Performs the forward pass through the network by applying the weights and biases to the input ‘x’, followed by the softmax activation function.

- ‘predict(self, X)’: Predicts class labels for given inputs ‘X’ by performing a forward pass and selecting the class with the highest probability.

- ‘accuracy(self, y\_true, y\_pred)’: Calculates the accuracy of predictions by comparing the predicted labels (‘y\_pred’) against the true labels (‘y\_true’).

- ‘train(self, X, y, epochs, lr)’: Trains the network using the training data (‘X’, ‘y’) for a specified number of epochs and learning rate (‘lr’). It implements stochastic gradient descent by iterating over individual samples, updating weights and biases based on the gradient of the loss. The method visualizes the loss and accuracy over epochs to monitor training progress.

Model Training and Evaluation

- Splitting the Data: The script utilizes ‘train\_test\_split’ from ‘sklearn.model\_selection’ to divide the data into training and testing sets, ensuring a portion of the data is reserved for evaluating the model's performance.

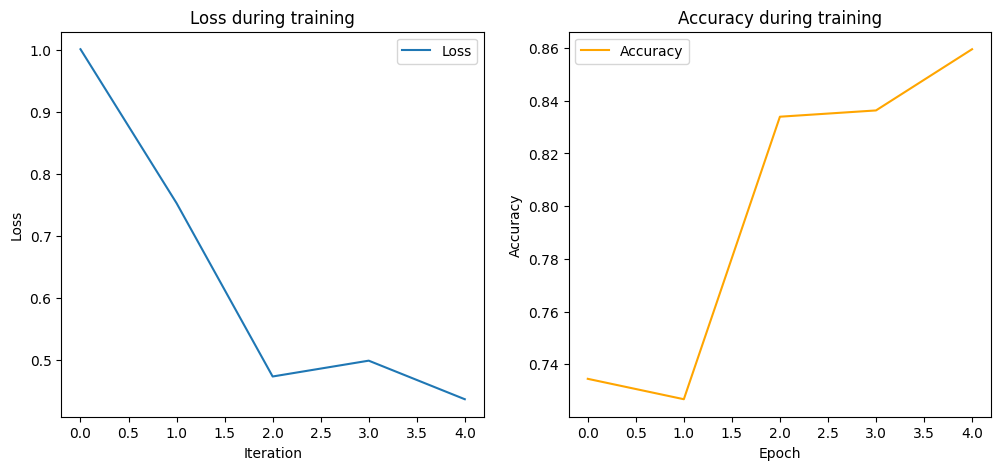
- Training the SLFN: An instance of the ‘SLFN’ class is created with specified input and output sizes, and the ‘train’ method is called to start the training process. This step involves forward propagation, loss computation, and parameter updates iteratively.

- Classification Report: After training, the model's predictive performance is evaluated using ‘classification\_report’ from ‘sklearn.metrics’ on both the training and test sets. This report includes metrics such as precision, recall, and F1-score, providing insights into the model's classification accuracy across different classes.

Result:

- Training the SLFN for a limited number of epochs demonstrated a steady decrease in loss and an increase in accuracy, indicating effective learning under the constraints.

- The classification reports for both training and test datasets revealed the model's performance across different classes, providing insights into its strengths and limitations in classifying various instances.

Loss – Accuracy plot:

Classification Report:

Train:

Classification Report:

precision recall f1-score support

0 0.97 0.96 0.96 278

1 0.94 0.97 0.95 270

2 0.80 0.94 0.86 290

3 0.74 0.75 0.75 279

4 0.87 0.79 0.83 279

5 0.86 0.75 0.80 284

accuracy 0.86 1680

macro avg 0.86 0.86 0.86 1680

weighted avg 0.86 0.86 0.86 1680

Test:

Classification Report:

precision recall f1-score support

0 0.99 0.99 0.99 72

1 0.87 0.93 0.90 80

2 0.79 0.98 0.87 60

3 0.79 0.73 0.76 71

4 0.85 0.85 0.85 71

5 0.94 0.73 0.82 66

accuracy 0.87 420

macro avg 0.87 0.87 0.86 420

weighted avg 0.87 0.87 0.86 420