Project LabelSOM

Coding group 06

- Emirhan Kurtulus 12243493
- Guilherme Monteiro Oliveira 12243299

Repository: https://github.com/GuiOli91/LabelSOM_group06

In [96]:
```
import numpy as np
import pandas as pdcoding
import gzip
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import json
```

In [97]:
```
#SOMToolbox Parser
from SOMToolBox_Parse import SOMToolBox_Parse
idata = SOMToolBox_Parse("datasets/iris/iris.vec").read_weight_file()
weights = SOMToolBox_Parse("datasets/iris/iris.wgt.gz").read_weight_file(
```

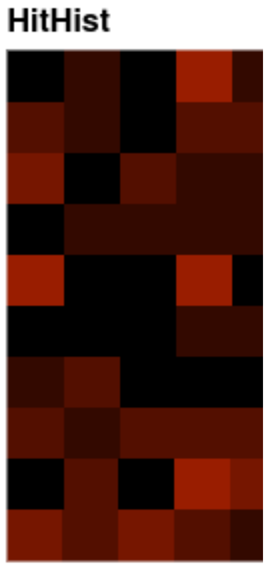
In [98]:
```
#HitHistogram
def HitHist(_m, _n, _weights, _idata):
    hist = np.zeros(_m * _n)
    for vector in _idata:
        position =np.argmin(np.sqrt(np.sum(np.power(_weights - vector, 2)
        hist[position] += 1

    return hist.reshape(_m, _n)

#U-Matrix - implementation
def UMatrix(_m, _n, _weights, _dim):
    U = _weights.reshape(_m, _n, _dim)
    U = np.insert(U, np.arange(1, _n), values=0, axis=1)
    U = np.insert(U, np.arange(1, _m), values=0, axis=0)
    #calculate interpolation
    for i in range(U.shape[0]):
        if i%2==0:
            for j in range(1,U.shape[1],2):
                U[i,j][0] = np.linalg.norm(U[i,j-1] - U[i,j+1], axis=-1)
        else:
            for j in range(U.shape[1]):
                if j%2==0:
                    U[i,j][0] = np.linalg.norm(U[i-1,j] - U[i+1,j], axis=
                else:
                    U[i,j][0] = (np.linalg.norm(U[i-1,j-1] - U[i+1,j+1],

    U = np.sum(U, axis=2) #move from Vector to Scalar

    for i in range(0, U.shape[0], 2): #count new values
        for j in range(0, U.shape[1], 2):
            region = []
            if j>0: region.append(U[i][j-1]) #check left border
            if i>0: region.append(U[i-1][j]) #check bottom
            if j<U.shape[1]-1: region.append(U[i][j+1]) #check right bord
```

```
                if i<U.shape[0]-1: region.append(U[i+1][j]) #check upper bord

                U[i,j] = np.median(region)

        return U

#SDH - implementation
def SDH(_m, _n, _weights, _idata, factor, approach):
    import heapq

    sdh_m = np.zeros( _m * _n)

    cs=0
    for i in range(factor): cs += factor-i

    for vector in _idata:
        dist = np.sqrt(np.sum(np.power(_weights - vector, 2), axis=1))
        c = heapq.nsmallest(factor, range(len(dist)), key=dist.__getitem_
        if (approach==0): # normalized
            for j in range(factor):  sdh_m[c[j]] += (factor-j)/cs
        if (approach==1):# based on distance
            for j in range(factor): sdh_m[c[j]] += 1.0/dist[c[j]]
        if (approach==2):
            dmin, dmax = min(dist[c]), max(dist[c])
            for j in range(factor): sdh_m[c[j]] += 1.0 - (dist[c[j]]-dmin

    return sdh_m.reshape(_m, _n)
```

In [99]:

```
import panel as pn
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')

hithist = hv.Image(HitHist(weights['ydim'], weights['ydim'], weights['arr
um = hv.Image(UMatrix(weights['ydim'], weights['ydim'], weights['arr'], 4
sdh = hv.Image(SDH(weights['ydim'], weights['ydim'], weights['arr'], idat

hv.Layout([hithist.relabel('HitHist').opts(cmap='kr'),
           um.relabel('U-Matrix').opts(cmap='jet'), sdh.relabel('SDH').op
```

Out[99]:



LabelSom

Task:

print the names of the n attributes A) per unit or B) per cluster as defined by one of the clustering techniques available that show 1) the lowest variance 2) the highest mean values within a specific unit or cluster, as well as 3) a weighted combination of these, as text on the map units / clusters.

Parameters that should be adjustable by the user include the maximum number of labels to be displayed, thresholds for the three selection types, relative weights for the combined selection of labels, the amount of detail being provided (labels-only or labels + values), and a grey-scaling or font size selection depending on the label weights (mean/variance/combined).

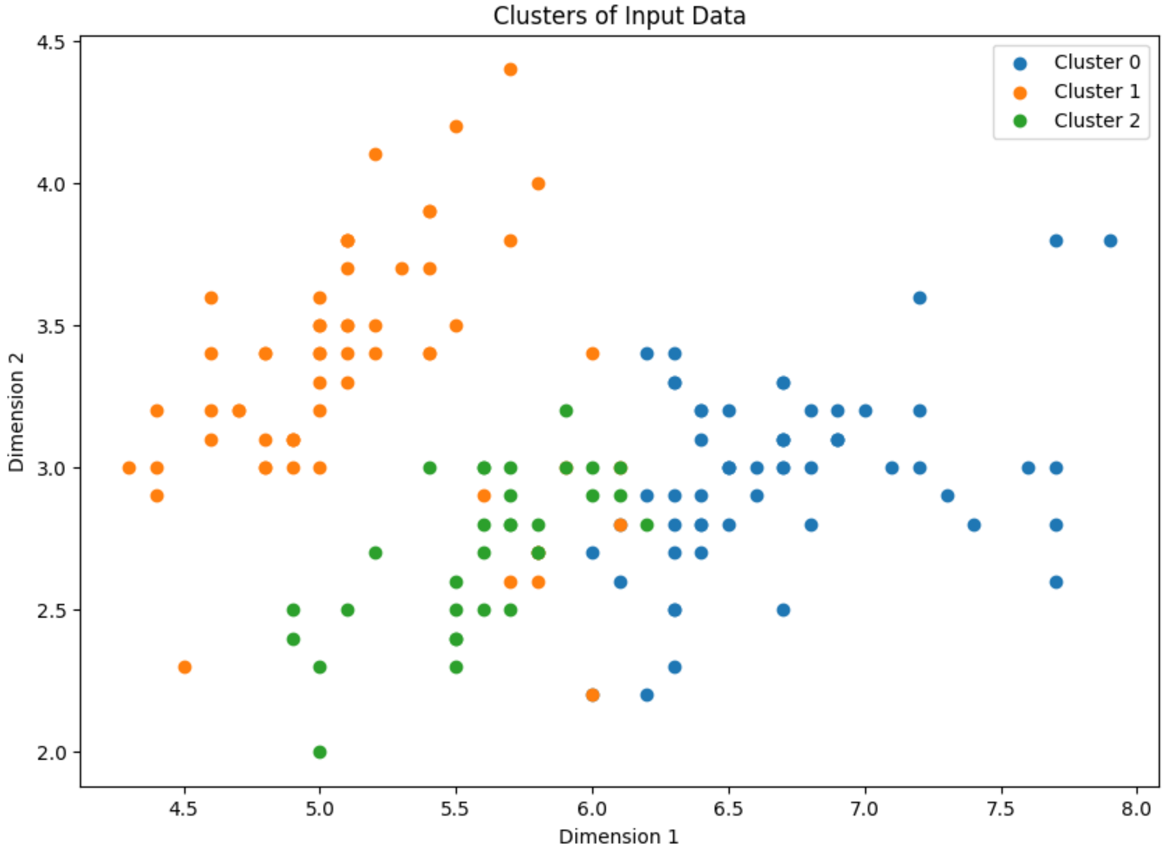
Solution:

By the given weights, we first reshape it to match the grid of the Self-Organizing Maps (SOM). Then, for each data instances we determine the closest weight to define the x, and y dimensions. In the following plot, we can see how the clusters are distribuited around the data instances. We can notice, based on the first and second attribute, Cluster two has an intersection with the others two clusters.

In [106…

```
# Reshape the weights to match the SOM grid
som_weights = weights['arr'].reshape((weights['xdim'], weights['ydim'], w

# Function to find the Best Matching Unit (BMU)
def find_bmu(input_vector, som_weights):
    xdim, ydim, vec_dim = som_weights.shape
    bmu_idx = None
    min_dist = float('inf')
    for x in range(xdim):
        for y in range(ydim):
            weight_vector = som_weights[x, y, :]
```

```
            dist = np.linalg.norm(input_vector - weight_vector)
            if dist < min_dist:
                min_dist = dist
                bmu_idx = (x, y)
    return bmu_idx

# Map each input vector to its BMU
bmu_indices = [find_bmu(vec, som_weights) for vec in idata['arr']]

# Convert BMU indices to a 2D array for clustering
bmu_array = np.array(bmu_indices)

# Perform K-means clustering with 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(bmu_array)
labels = kmeans.labels_

# Assign each input vector to the cluster of its corresponding BMU
idata_labels = labels

# Plot the clusters
plt.figure(figsize=(10, 7))

# Use the first two dimensions for plotting
for cluster in range(3):
    cluster_points = idata['arr'][idata_labels == cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Clust

plt.title('Clusters of Input Data')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```



LabelSOM Class

The `LabelSOM` class is designed to visualize labels on a Self-Organizing Map (SOM) corresponding to the most significant attributes for each cluster or unit. It allows user-adjustable parameters such as the maximum number of labels to display, thresholds for selecting labels based on mean values. The class includes methods to compute cluster statistics, where the mean feature values of each cluster are calculated, and a method to cluster SOM units using K-Means, from scikit learning.

Methods:

- `compute_cluster_stats(clusters)` : Computes the mean values for each cluster.
- `cluster_som_units(n_clusters)` : Clusters the SOM units using KMeans.
- `save_provenance(method_name, params)` : Saves provenance information for method calls.
- `plot_labels(n_clusters)` : Plots the SOM clusters with labels for the highest mean point in each cluster.

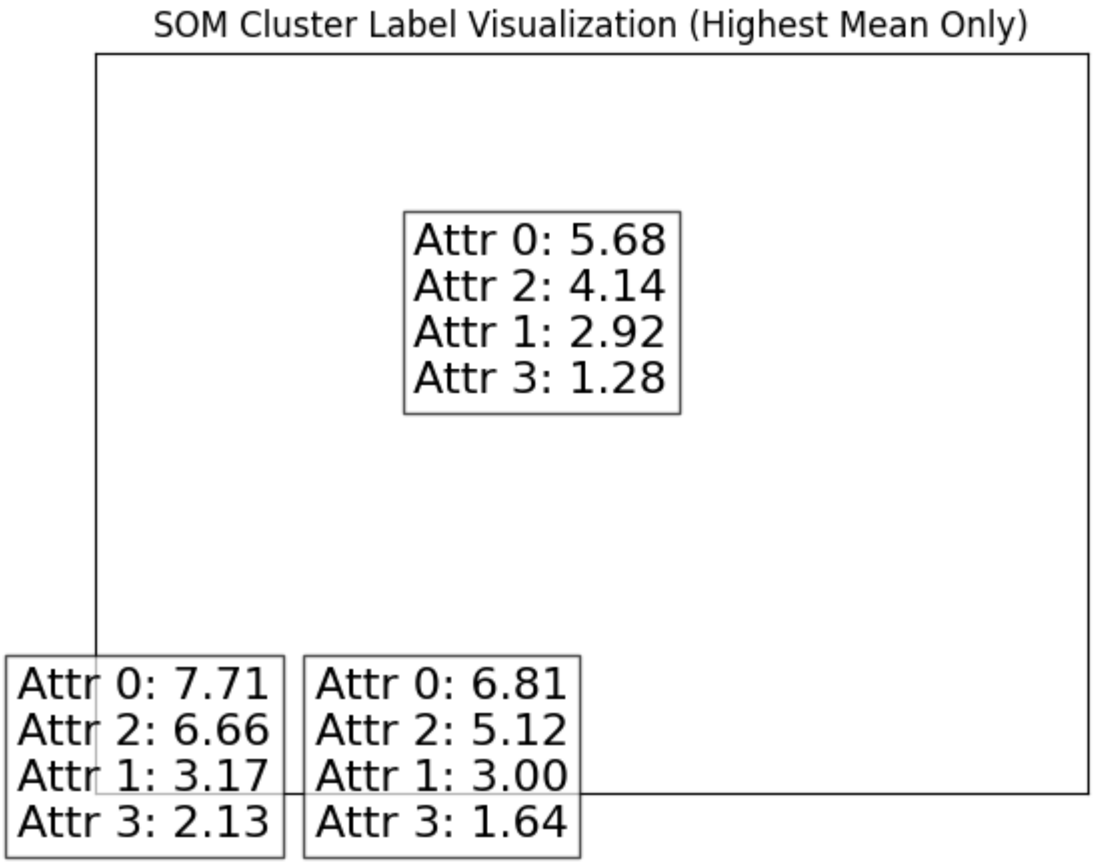
In [104…

```
class LabelSOM:
    def __init__(self, som_map, data, labels, max_labels=4, mean_threshol
        self.som_map = som_map  # Trained SOM units
        self.data = data  # Data mapped to SOM
        self.labels = labels  # Feature labels
        self.max_labels = max_labels
        self.mean_threshold = mean_threshold
        self.display_mode = display_mode
        self.font_scaling = font_scaling

    def compute_cluster_stats(self, clusters):
        """Compute mean per cluster"""
        cluster_stats = {}
        for cluster_id, units in clusters.items():
            cluster_data = np.vstack([self.som_map[unit] for unit in unit
            if len(cluster_data) > 0:
                mean_values = np.mean(cluster_data, axis=0)
                cluster_stats[cluster_id] = mean_values
        return cluster_stats

    def cluster_som_units(self, n_clusters=3):
        """Cluster SOM units using KMeans"""
        unit_positions = np.array(list(self.som_map.keys()))
        kmeans = KMeans(n_clusters=n_clusters, random_state=42).fit(unit_
        clusters = {i: [] for i in range(n_clusters)}
        for i, label in enumerate(kmeans.labels_):
            clusters[label].append(tuple(unit_positions[i]))
        return clusters

    def save_provenance(self, method_name, params):
        """Save provenance information for method calls"""
        provenance = {
            "method": method_name,
            "parameters": params
        }
        with open("provenance.json", "a") as f:
```

```
            f.write(json.dumps(provenance) + "\n")

    def plot_labels(self, n_clusters=3):
        """Plot SOM clusters with labels for the highest mean point in ea
        self.save_provenance("plot_labels", {"n_clusters": n_clusters})
        clusters = self.cluster_som_units(n_clusters=n_clusters)
        cluster_stats = self.compute_cluster_stats(clusters)

        fig, ax = plt.subplots()
        for cid, means in cluster_stats.items():
            # Find the unit with the highest mean value in the cluster
            highest_mean_unit = max(
                clusters[cid],
                key=lambda unit: np.mean(self.som_map[unit]) if unit in s
            )

            if highest_mean_unit in self.som_map:
                means = self.som_map[highest_mean_unit]
                top_labels = sorted(zip(self.labels, means), key=lambda x
                text = '\n'.join(
                    [f"{lbl}: {val:.2f}" if self.display_mode == 'labels+
                )
                # Display label only for the highest mean point
                ax.text(
                    highest_mean_unit[0], highest_mean_unit[1], text, ha=
                    fontsize=16 if self.font_scaling else 10,
                    bbox=dict(facecolor='white', alpha=0.7)
                )

        grid_x, grid_y = zip(*self.som_map.keys())
        ax.set_xlim(min(grid_x) - 0.5, max(grid_x) + 0.5)
        ax.set_ylim(min(grid_y) - 0.5, max(grid_y) + 0.5)
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title("SOM Cluster Label Visualization (Highest Mean Only)
        plt.gca().invert_yaxis()
        plt.show()
```

In [105…

```
som_weights = weights['arr'].reshape((weights['xdim'], weights['ydim'], w
som_map = { (x,y): som_weights[x,y,:] for x in range(10) for y in range(1
labels = [f'Attr {i}' for i in range(5)]
label_som = LabelSOM(som_map, idata["arr"], labels)
label_som.plot_labels()
```


SOM Cluster Label Visualization (Highest Mean Only)
Attr 0: 5.68
Attr 2: 4.14
Attr 1: 2.92
Attr 3: 1.28
Attr 0: 7.71
Attr 2: 6.66
Attr 1: 3.17
Attr 3: 2.13
Attr 0: 6.81
Attr 2: 5.12
Attr 1: 3.00
Attr 3: 1.64