CO542- NEURAL NETWORKS AND FUZZY SYSTEMS

SELF ORGANIZING MAPS (SOM)

F.S.MARZOOK

E/16/232

LAB 05

17/10/2021

CODE

```
import pandas as pd
from matplotlib.lines import Line2D
from sklearn.preprocessing import MinMaxScaler
from minisom import MiniSom
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

# 1. Load the data set as a pandas' data frame and generate the feature matrix by selecting the
# appropriate features in the data set.
diabetes_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/diabetes.csv')

x_col = [col for col in diabetes_data.columns if col not in ['Outcome']]
y_col = 'Outcome'
features = diabetes_data[x_col]
features_val = features.values
target = diabetes_data[y_col]
target_val = target.values
# print(target_val)

# 2. Use sklearn.preprocessing.MinMaxScaler normalize the data between 0 and 1.
minMaxScalar = MinMaxScaler()
features_scaled = minMaxScalar.fit_transform(features_val)
# print(features_scaled)

# 3. Initialize the weights using MiniSom.pca_weights_init function (Also identify the usage of
# random_weights_init method and mention an advantage of using pca_weights_init).
som = MiniSom(6, 6, 8, sigma=0.3, learning_rate=0.5)           # initialization of 6x6 SOM
som.pca_weights_init(features_scaled)                          # Initialize the weights

# 4. Identify the difference between Minisom.train_batch and Minisom.train_random methods available
# minisom and use one of the models to train your self-organizing map.
som.train_random(features_scaled, 5000)                        # trains the SOM with 5000 iterations
```

```
# 6. Use Minisom.distance_map function to visualize the results obtained from the training process and
# use appropriate markers to indicate individual samples matched into each cell (according to the
# classes; patient who has diabetes/ do not have diabetes define two markers).
plt.figure(figsize=(8, 7))
plt.pcolor(som.distance_map().T, cmap='viridis')
plt.colorbar()
markers = ['o', 'x']
colors = ['C0', 'C1']
for id, x in enumerate(features_scaled):
    w = som.winner(x)          # getting the winner
    # place a marker on the winning position for the sample x
    plt.plot(w[0] + 0.5, w[1] + 0.5,
             markers[target_val[id] - 1],
             markerfacecolor='None',
             markeredgecolor=colors[target_val[id] - 1],
             markersize=12,
             markeredgewidth=2)
legend_elements = [Line2D([0], [0], marker=markers[0], color=colors[0], label='Has Diabetes',
                          markerfacecolor='w', markersize=12, linestyle='None', markeredgewidth=2),
                   Line2D([0], [0], marker=markers[1], color=colors[1], label='Do not have Diabetes',
                          markerfacecolor='w', markersize=12, linestyle='None', markeredgewidth=2)]
plt.legend(handles=legend_elements, bbox_to_anchor=(0.05, 1.1), loc='upper left',
           borderaxespad=0., ncol=3, fontsize=12)
plt.show()
```

```
# 8. Visualize the proportion of samples per class falling in a specific neuron using the,
# matplotlib.gridspec.GridSpec and matplotlib.patches.Patcha
labels_map = som.labels_map(features_scaled, target_val)
fig = plt.figure(figsize=(6, 6))
the_grid = gridspec.GridSpec(6, 6, fig)
for position in labels_map.keys():
    label_fracs = [labels_map[position][t] for t in target.unique()]
    plt.subplot(the_grid[6-1-position[1], position[0]], aspect=1)
    patches, texts = plt.pie(label_fracs)

plt.legend(target.unique(), bbox_to_anchor=(0, 0), ncol=3, loc='upper left', borderaxespad=0.)
plt.show()
```

3. Usage of random_weights_init method:
   It initializes the weights of the Self-Organizing Maps by picking random samples from the data.

   an advantage of using pca_weights_init method:
   It does not depend on the random processes and converges the training process faster.

4. train_random method is the random training, where the model is trained by picking random samples from the data
   train_batch is the batch training, where the model is trained by picking the samples in the order they are stored.

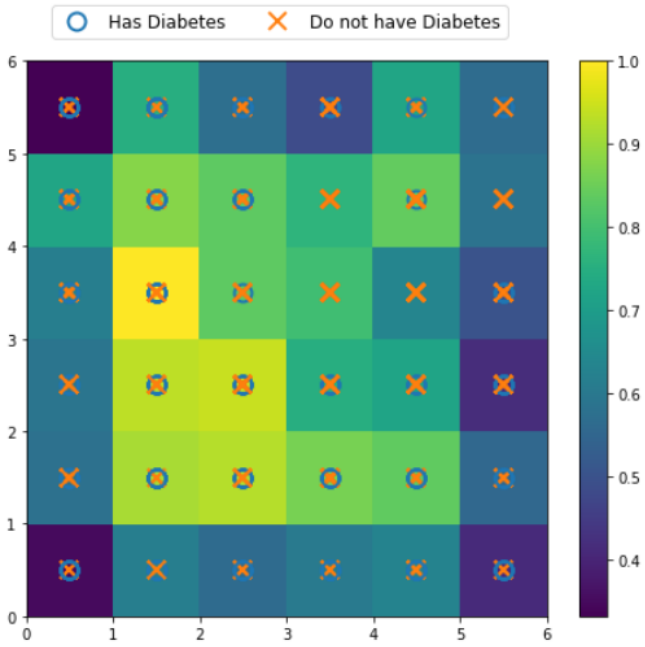
5. Each cell in the distance map is the normalized sum of Euclidean distances between a neuron and its neighbors.
6. 

Figure 1: Results Obtained from the Training Process

7.

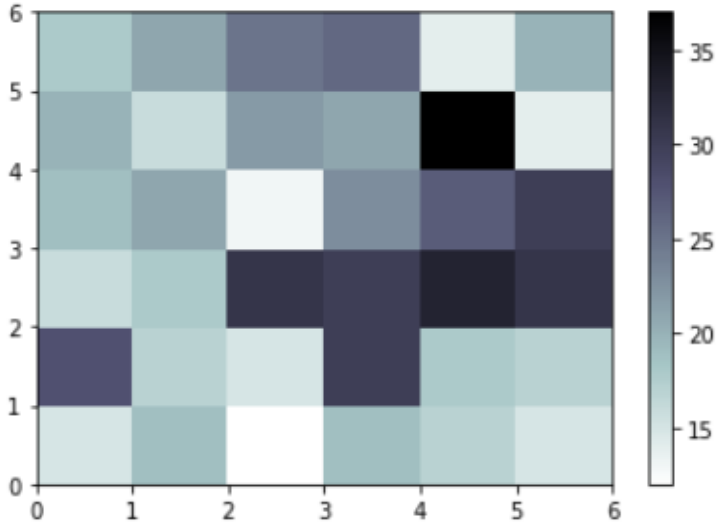


Figure 2: Activation Response

8.

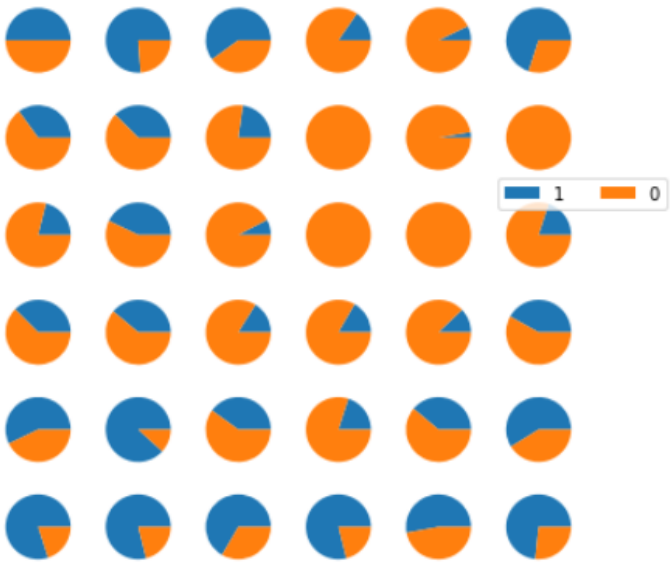


Figure 3: Proportion of Samples per Class Falling in a Specific Neuron