Self Organizing Map

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Importing the dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
CustomerID    690 non-null int64
A1            690 non-null int64
A2            690 non-null float64
A3            690 non-null float64
A4            690 non-null int64
A5            690 non-null int64
A6            690 non-null int64
A7            690 non-null float64
A8            690 non-null int64
A9            690 non-null int64
A10           690 non-null int64
A11           690 non-null int64
A12           690 non-null int64
A13           690 non-null int64
A14           690 non-null int64
Class         690 non-null int64
dtypes: float64(3), int64(13)
memory usage: 86.3 KB
```

In [3]:
```
dataset.head()
```

Out[3]:

	CustomerID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	Class
0	15776156	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	15739548	0	22.67	7.00	2	8	4	0.165	0	0	0	0	2	160	1	0
2	15662854	0	29.58	1.75	1	4	4	1.250	0	0	0	1	2	280	1	0
3	15687688	0	21.67	11.50	1	5	3	0.000	1	1	11	1	2	0	1	1
4	15715750	1	20.17	8.17	2	6	4	1.960	1	1	14	0	2	60	159	1

In [4]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [5]:
```
X
```

Out[5]:
```
array([[1.5776156e+07, 1.0000000e+00, 2.2080000e+01, ..., 2.0000000e+00,
        1.0000000e+02, 1.2130000e+03],
       [1.5739548e+07, 0.0000000e+00, 2.2670000e+01, ..., 2.0000000e+00,
        1.6000000e+02, 1.0000000e+00],
       [1.5662854e+07, 0.0000000e+00, 2.9580000e+01, ..., 2.0000000e+00,
        2.8000000e+02, 1.0000000e+00],
       ...,
       [1.5675450e+07, 0.0000000e+00, 1.8830000e+01, ..., 2.0000000e+00,
        1.0000000e+02, 1.0000000e+00],
       [1.5776494e+07, 0.0000000e+00, 2.7420000e+01, ..., 2.0000000e+00,
        1.2000000e+02, 1.2000000e+01],
       [1.5592412e+07, 1.0000000e+00, 4.1000000e+01, ..., 1.0000000e+00,
        5.6000000e+02, 1.0000000e+00]])
```

In [6]:
```
y
```

Out[6]:
```
array([0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 1, 1], dtype=int64)
```

In [7]:
```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)
```

In [8]:
```
X[10]
```

Out[8]:
```
array([0.31526975, 1.        , 0.29699248, 0.0625    , 0.5       ,
       1.        , 0.875     , 0.15789474, 1.        , 1.        ,
       0.05970149, 1.        , 0.5       , 0.1265    , 0.00857   ])
```

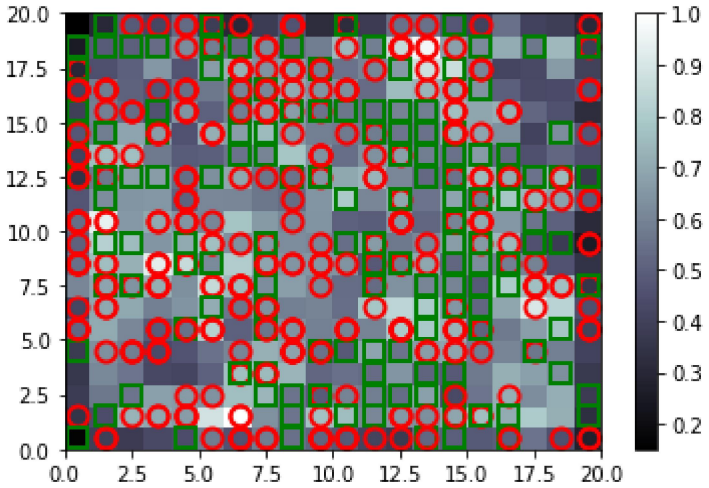
In [9]:
```
# Training the SOM
from minisom_new import MiniSom
som = MiniSom(x = 20, y = 20, input_len = 15, sigma = 1.0, learning_rate = 0.2)
'''
Parameters
        ----------
        x : int
            x dimension of the SOM.

        y : int
            y dimension of the SOM.

        input_len : int
            Number of the elements of the vectors in input.

        sigma : float, optional (default=1.0)
            Spread of the neighborhood function, needs to be adequate
            to the dimensions of the map.
            (at the iteration t we have sigma(t) = sigma / (1 + t/T)
            where T is #num_iteration/2)
        learning_rate, initial learning rate
            (at the iteration t we have
            learning_rate(t) = learning_rate / (1 + t/T)
            where T is #num_iteration/2)
        random_seed : int, optional (default=None)
            Random seed to use.
'''
som.random_weights_init(X)
"""Initializes the weights of the SOM
        picking random samples from data"""
som.train_random(data = X, num_iteration = 100)
# """Trains the SOM picking samples at random from data"""
```

In [11]:
```
# Visualizing the results
from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
"""Returns the distance map of the weights.
        Each cell is the normalised sum of the distances between
        a neuron and its neighbours."""
colorbar()
markers = ['o', 's']
colors = ['r', 'g']
for i, x in enumerate(X):
    w = som.winner(x) # """Computes the coordinates of the winning neuron for the
    plot(w[0] + 0.5,
         w[1] + 0.5,
         markers[y[i]],
         markeredgecolor = colors[y[i]],
         markerfacecolor = 'None',
         markersize = 10,
         markeredgewidth = 2)
show()
```



In [12]:
```
mappings = som.win_map(X)
"""Returns a dictionary wm where wm[(i,j)] is a list
        with all the patterns that have been mapped in the position i,j."""
```

In [13]:
```
mappings
```

Out[13]:
```
defaultdict(list,
            {(9,
              4): [array([0.84268147, 1.        , 0.12526316, 0.40928571, 0.5
       ,
                          0.23076923, 0.375     , 0.05561404, 0.        , 0.
  ,
                          0.        , 1.        , 0.5       , 0.05      , 0.01212
  ]), array([0.59042402, 1.        , 0.20556391, 0.44642857, 0.5       ,
                          0.38461538, 0.5       , 0.00877193, 0.        , 0.
  ,
                          0.        , 1.        , 0.5       , 0.36      , 0.
  ]), array([0.89857005, 1.        , 0.05639098, 0.78571429, 1.        ,
                          0.        , 0.75      , 0.        , 0.        , 0.
  ,
                          0.        , 1.        , 1.        , 0.225     , 1.
  ]), array([0.67920025, 1.        , 0.19428571, 0.52089286, 0.5       ,
                          0.15384615, 0.5       , 0.        , 0.        , 0.
  ,
                          0.        , 1.        , 0.5       , 0.089     , 0.
```

In [17]:
```
labels_maps = som.labels_map(X, y)
"""Returns a dictionary wm where wm[(i,j)] is a dictionary
       that contains the number of samples from a given label
       that have been mapped in position i,j.

       Parameters
       ----------
       data : data matrix

       label : list or array that contains the label of each sample in data.
       """
```

In [18]:
```
labels_maps
```

Out[18]:
```
defaultdict(list,
            {(9, 4): Counter({0: 3, 1: 1}),
             (7, 6): Counter({0: 5, 1: 1}),
             (8, 4): Counter({0: 5}),
             (12, 15): Counter({1: 5}),
             (17, 10): Counter({1: 1}),
             (18, 5): Counter({1: 5}),
             (3, 5): Counter({0: 3}),
             (8, 0): Counter({1: 4}),
             (5, 14): Counter({0: 3}),
             (0, 1): Counter({0: 1}),
             (14, 3): Counter({1: 4}),
             (19, 12): Counter({1: 4}),
             (16, 8): Counter({0: 1, 1: 1}),
             (12, 4): Counter({1: 1}),
             (14, 14): Counter({0: 4, 1: 1}),
             (2, 12): Counter({1: 1}),
             (15, 7): Counter({1: 2}),
             (0, 14): Counter({1: 6, 0: 1}),
```

In [ ]: