Self-Organizing Maps for Fraud Detection

This project involves detecting fraud in credit card applications. We will use Unsupervised Deep Learning to group different customers based on patterns and then get the group that resembles fraudulent.

In a broad sense, the fraud will consist of the outliers, simply because fraud encompasses something which does not follow the general rule. And so, unlike the ordinary rows, which will be bundled together, the fraud data points will be far beyond.

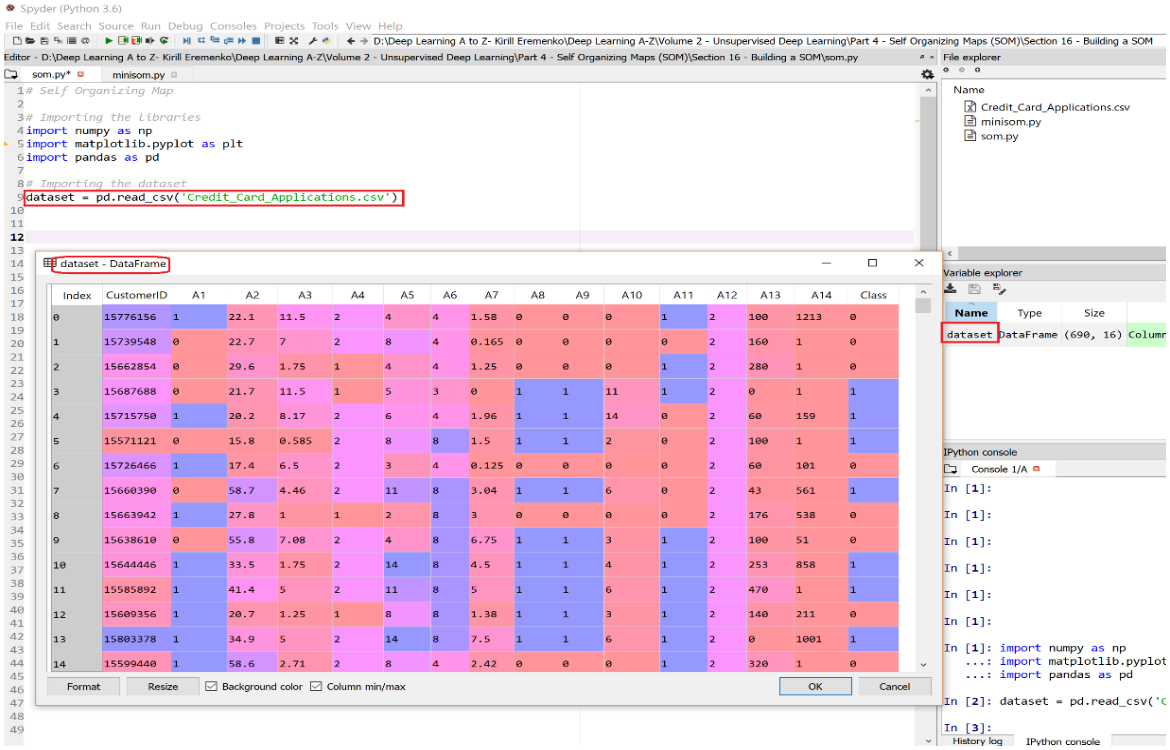
About SOM

Self-Organizing Map (SOM) is designed to get data from a table with large number of rows and columns and convert it into a 2-d map of sorts, thus reducing its dimensionality.

It is an Unsupervised learning algorithm i.e. it does not require labels for classification of the data. It simply 'groups' the data based on common attributes. By grouping data that is similar, we can get rid of additional columns which are not easily understood by humans.

Importing Data

The first step is importing the right libraries and later, the credit card applications dataset. The dataset is taken from the UCI Data Repository. The dataset we are currently working on is the Statlog (Australian Credit Approval) dataset. All attribute names and values have been changed to meaningless values, making it impossible for humans to get any patterns, and therefore making unsupervised deep learning model necessary.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
Credit_Card_Applications.csv
minisom.py
som.py
dataset - DataFrame
Format
Resize
☑ Background color
☑ Column min/max
OK
Cancel
Variable explorer
IPython console
Console 1/A
History log
IPython console

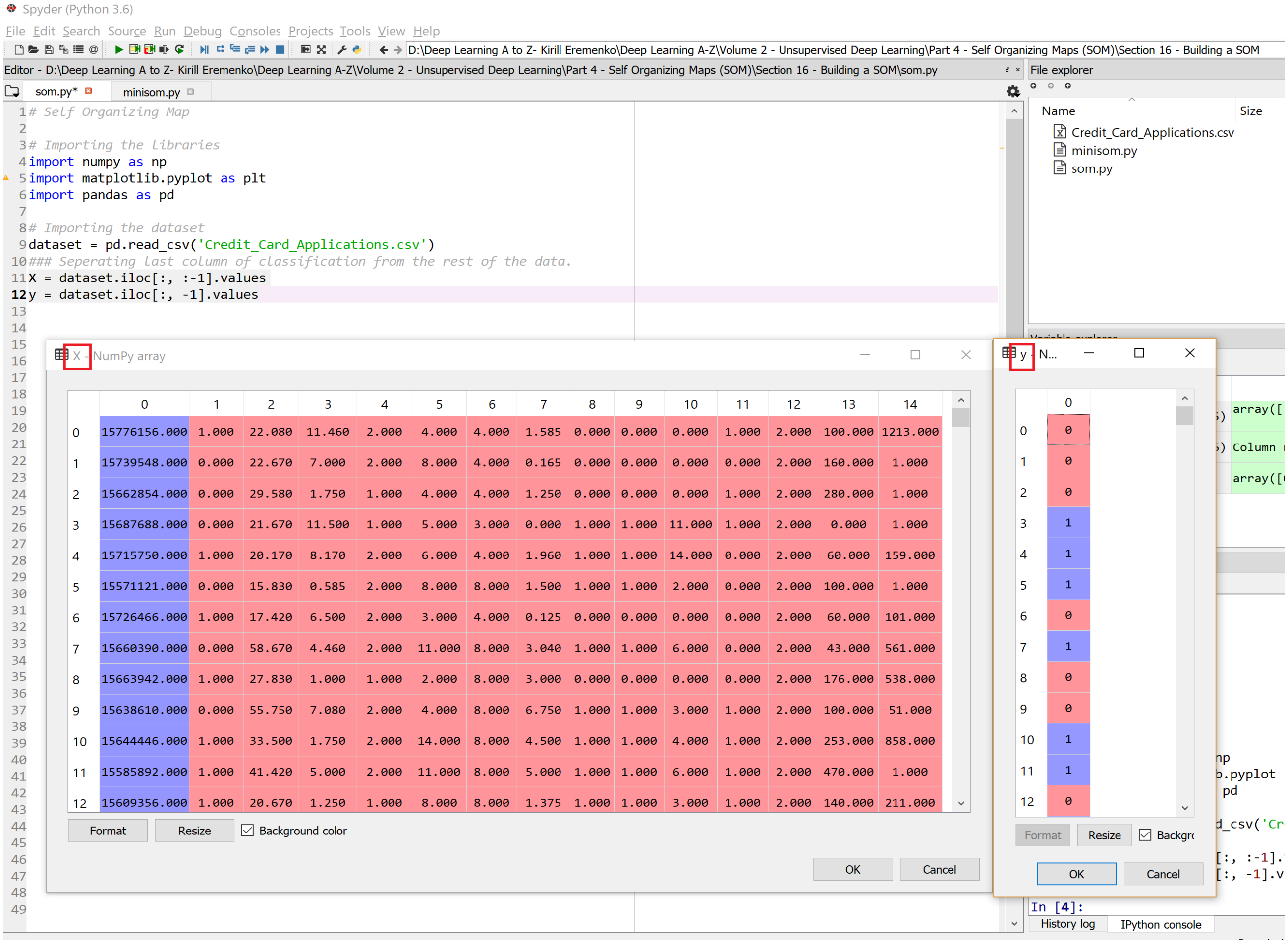
```
# Self Organizing Map

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
```

Index	CustomerID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	Class
0	15776156	1	22.1	11.5	2	4	4	1.58	0	0	0	1	2	100	1213	0
1	15739548	0	22.7	7	2	8	4	0.165	0	0	0	0	2	160	1	0
2	15662854	0	29.6	1.75	1	4	4	1.25	0	0	0	1	2	280	1	0
3	15687688	0	21.7	11.5	1	5	3	0	1	1	11	1	2	0	1	1
4	15715750	1	20.2	8.17	2	6	4	1.96	1	1	14	0	2	60	159	1
5	15571121	0	15.8	0.585	2	8	8	1.5	1	1	2	0	2	100	1	1
6	15726466	1	17.4	6.5	2	3	4	0.125	0	0	0	0	2	60	101	0
7	15660390	0	58.7	4.46	2	11	8	3.04	1	1	6	0	2	43	561	1
8	15663942	1	27.8	1	1	2	8	3	0	0	0	0	2	176	538	0
9	15638610	0	55.8	7.08	2	4	8	6.75	1	1	3	1	2	100	51	0
10	15644446	1	33.5	1.75	2	14	8	4.5	1	1	4	1	2	253	858	1
11	15585892	1	41.4	5	2	11	8	5	1	1	6	1	2	470	1	1
12	15609356	1	20.7	1.25	1	8	8	1.38	1	1	3	1	2	140	211	0
13	15803378	1	34.9	5	2	14	8	7.5	1	1	6	1	2	0	1001	1
14	15599440	1	58.6	2.71	2	8	4	2.42	0	0	0	1	2	320	1	0

Name	Type	Size	
dataset	DataFrame	(690, 16)	Columr

```
In [1]:

In [1]:

In [1]:

In [1]:

In [1]:

In [1]:

In [1]: import numpy as np
   ...: import matplotlib.pyplot
   ...: import pandas as pd

In [2]: dataset = pd.read_csv('C

In [3]:
```

Every row is a customer and the columns represent their attributes. The final column (Class) contains the data whether the application was approved or rejected. So, we will separate this column from the rest of the dataset. This will help us in making this dataset "fit for unsupervised learning" and help us in verifying if our predictions are true or not. Please note that we are not splitting the data because we want to do supervised learning. During our training we will only be using X and not y part of the dataset.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
Size
Credit_Card_Applications.csv
minisom.py
som.py
In [4]:
History log
IPython console

```python
# Self Organizing Map

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
### Seperating last column of classification from the rest of the data.
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

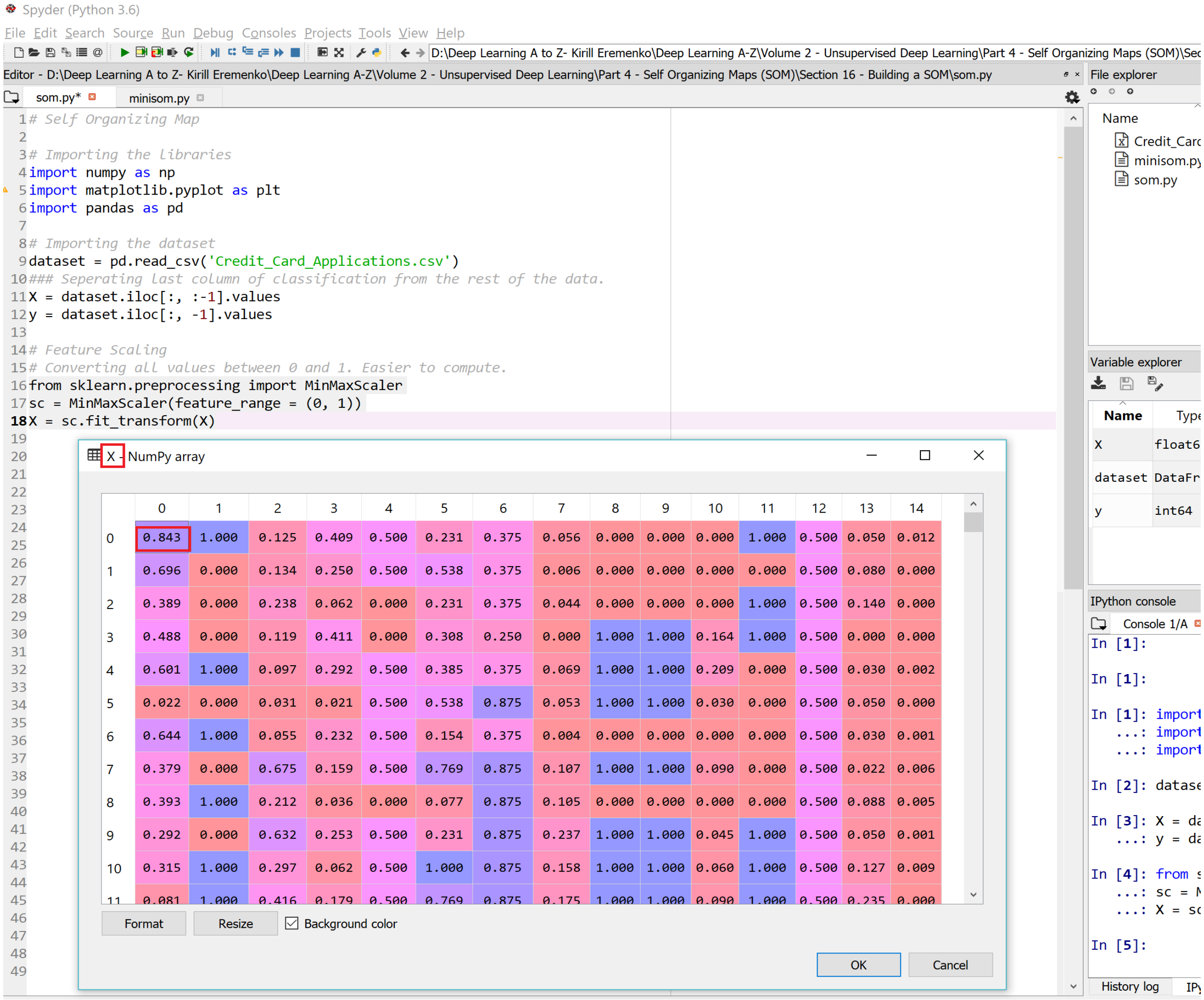
X - NumPy array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	15776156.000	1.000	22.080	11.460	2.000	4.000	4.000	1.585	0.000	0.000	0.000	1.000	2.000	100.000	1213.000
1	15739548.000	0.000	22.670	7.000	2.000	8.000	4.000	0.165	0.000	0.000	0.000	0.000	2.000	160.000	1.000
2	15662854.000	0.000	29.580	1.750	1.000	4.000	4.000	1.250	0.000	0.000	0.000	1.000	2.000	280.000	1.000
3	15687688.000	0.000	21.670	11.500	1.000	5.000	3.000	0.000	1.000	1.000	11.000	1.000	2.000	0.000	1.000
4	15715750.000	1.000	20.170	8.170	2.000	6.000	4.000	1.960	1.000	1.000	14.000	0.000	2.000	60.000	159.000
5	15571121.000	0.000	15.830	0.585	2.000	8.000	8.000	1.500	1.000	1.000	2.000	0.000	2.000	100.000	1.000
6	15726466.000	1.000	17.420	6.500	2.000	3.000	4.000	0.125	0.000	0.000	0.000	0.000	2.000	60.000	101.000
7	15660390.000	0.000	58.670	4.460	2.000	11.000	8.000	3.040	1.000	1.000	6.000	0.000	2.000	43.000	561.000
8	15663942.000	1.000	27.830	1.000	1.000	2.000	8.000	3.000	0.000	0.000	0.000	0.000	2.000	176.000	538.000
9	15638610.000	0.000	55.750	7.080	2.000	4.000	8.000	6.750	1.000	1.000	3.000	1.000	2.000	100.000	51.000
10	15644446.000	1.000	33.500	1.750	2.000	14.000	8.000	4.500	1.000	1.000	4.000	1.000	2.000	253.000	858.000
11	15585892.000	1.000	41.420	5.000	2.000	11.000	8.000	5.000	1.000	1.000	6.000	1.000	2.000	470.000	1.000
12	15609356.000	1.000	20.670	1.250	1.000	8.000	8.000	1.375	1.000	1.000	3.000	1.000	2.000	140.000	211.000

y - N...

	0
0	0
1	0
2	0
3	1
4	1
5	1
6	0
7	1
8	0
9	0
10	1
11	1
12	0

We need Feature Scaling as it helps deal with multi-dimensional datasets and makes computations easier. We will use Normalization to convert all our values between the range of 0 and 1. And we will convert using the MinMaxScaler from sklearn to do so. We then fit our input data (X) into the scaling object and transform it.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Sec
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
Credit_Carc
minisom.py
som.py
Variable explorer
IPython console
Console 1/A
History log
IPy

```
# Self Organizing Map

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Credit_Card_Applications.csv')
### Seperating last column of classification from the rest of the data.
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Feature Scaling
# Converting all values between 0 and 1. Easier to compute.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)
```

X - NumPy array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.843	1.000	0.125	0.409	0.500	0.231	0.375	0.056	0.000	0.000	0.000	1.000	0.500	0.050	0.012
1	0.696	0.000	0.134	0.250	0.500	0.538	0.375	0.006	0.000	0.000	0.000	0.000	0.500	0.080	0.000
2	0.389	0.000	0.238	0.062	0.000	0.231	0.375	0.044	0.000	0.000	0.000	1.000	0.500	0.140	0.000
3	0.488	0.000	0.119	0.411	0.000	0.308	0.250	0.000	1.000	1.000	0.164	1.000	0.500	0.000	0.000
4	0.601	1.000	0.097	0.292	0.500	0.385	0.375	0.069	1.000	1.000	0.209	0.000	0.500	0.030	0.002
5	0.022	0.000	0.031	0.021	0.500	0.538	0.875	0.053	1.000	1.000	0.030	0.000	0.500	0.050	0.000
6	0.644	1.000	0.055	0.232	0.500	0.154	0.375	0.004	0.000	0.000	0.000	0.000	0.500	0.030	0.001
7	0.379	0.000	0.675	0.159	0.500	0.769	0.875	0.107	1.000	1.000	0.090	0.000	0.500	0.022	0.006
8	0.393	1.000	0.212	0.036	0.000	0.077	0.875	0.105	0.000	0.000	0.000	0.000	0.500	0.088	0.005
9	0.292	0.000	0.632	0.253	0.500	0.231	0.875	0.237	1.000	1.000	0.045	1.000	0.500	0.050	0.001
10	0.315	1.000	0.297	0.062	0.500	1.000	0.875	0.158	1.000	1.000	0.060	1.000	0.500	0.127	0.009
11	0.081	1.000	0.416	0.179	0.500	0.769	0.875	0.175	1.000	1.000	0.090	1.000	0.500	0.235	0.000

Format | Resize | ☑ Background color

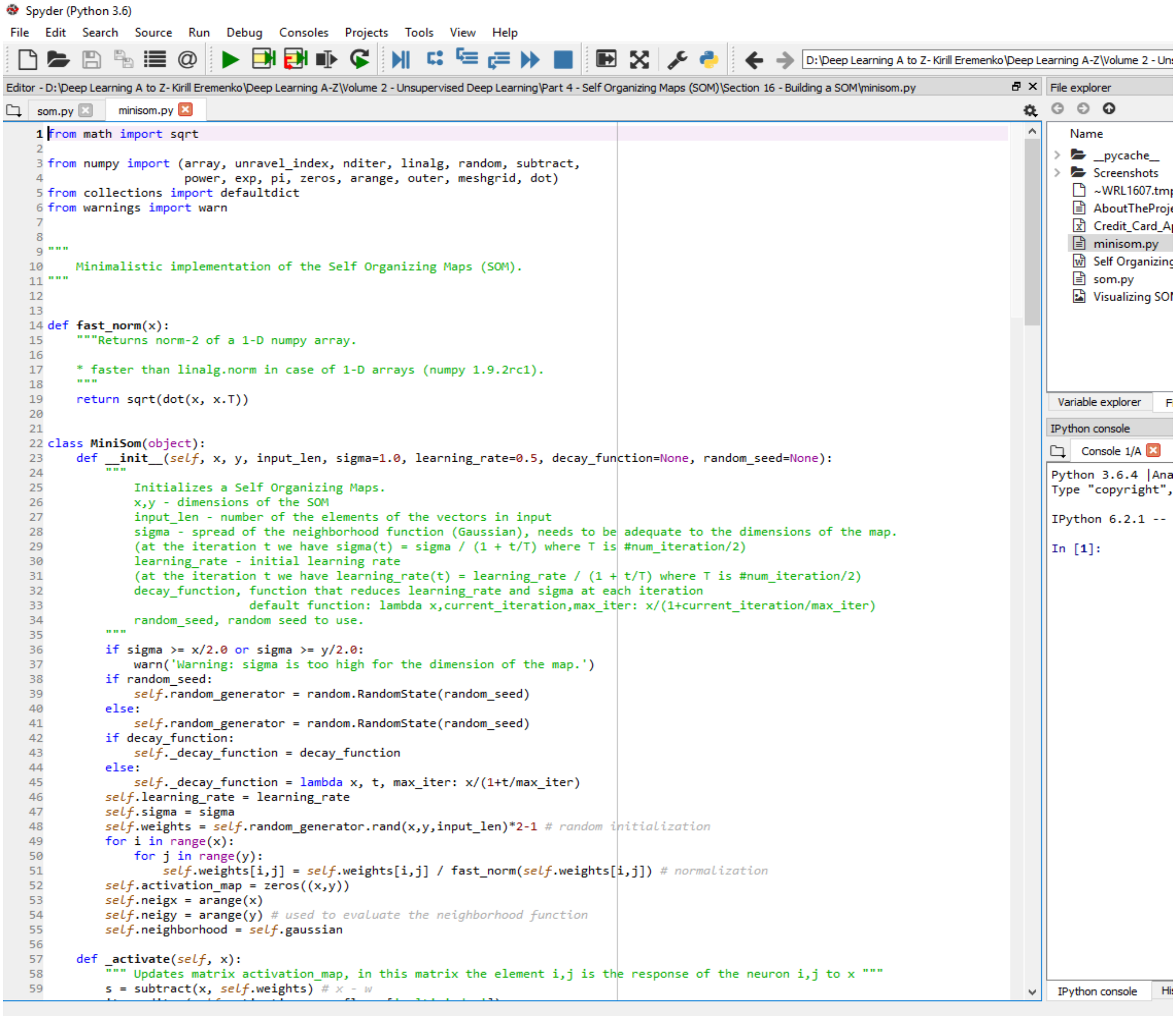
OK | Cancel

Name	Type
X	float6
dataset	DataFr
y	int64

```
In [1]:

In [1]:

In [1]: import
   ...: import
   ...: import

In [2]: datase

In [3]: X = da
   ...: y = da

In [4]: from s
   ...: sc = M
   ...: X = sc

In [5]:
```

Training the SOM

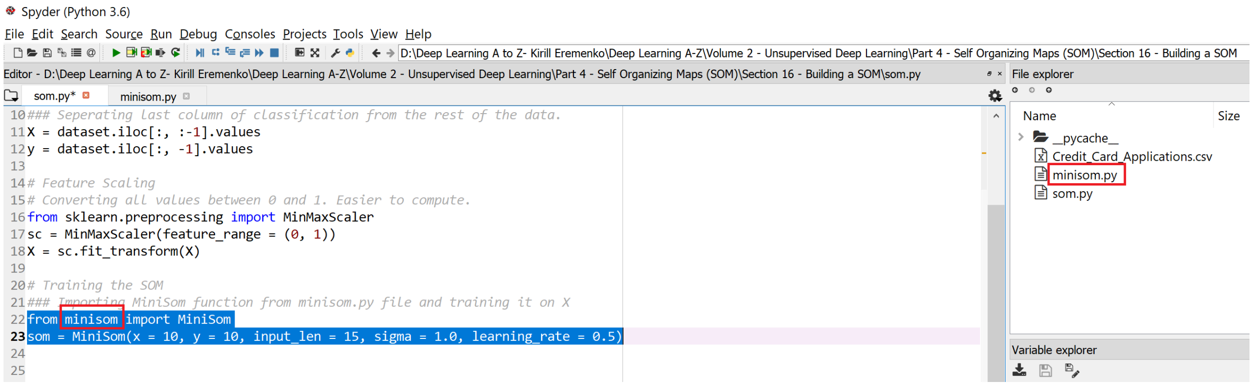
There are two ways to implement SOM - by creating your own code for scratch, or to use an existing implementation which works well, just like scikit-learn. The implementation we will be using is Minisom 1.0 (https://pypi.python.org/pypi/MiniSom/1.0). Its license (CC by 3.0) allows us to use it freely, which is another reason favoring this implementation. The implementation consists of a Python code (minisom.py) in our working directory, which we will be incorporating in our code.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Un
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\minisom.py
som.py
minisom.py
File explorer
Name
__pycache__
Screenshots
~WRL1607.tmp
AboutTheProje
Credit_Card_A
minisom.py
Self Organizing
som.py
Visualizing SOI
Variable explorer
IPython console
Console 1/A
IPython console

```python
from math import sqrt

from numpy import (array, unravel_index, nditer, linalg, random, subtract,
                   power, exp, pi, zeros, arange, outer, meshgrid, dot)
from collections import defaultdict
from warnings import warn


"""
    Minimalistic implementation of the Self Organizing Maps (SOM).
"""


def fast_norm(x):
    """Returns norm-2 of a 1-D numpy array.

    * faster than linalg.norm in case of 1-D arrays (numpy 1.9.2rc1).
    """
    return sqrt(dot(x, x.T))


class MiniSom(object):
    def __init__(self, x, y, input_len, sigma=1.0, learning_rate=0.5, decay_function=None, random_seed=None):
        """
            Initializes a Self Organizing Maps.
            x,y - dimensions of the SOM
            input_len - number of the elements of the vectors in input
            sigma - spread of the neighborhood function (Gaussian), 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)
            decay_function, function that reduces learning_rate and sigma at each iteration
                            default function: lambda x,current_iteration,max_iter: x/(1+current_iteration/max_iter)
            random_seed, random seed to use.
        """
        if sigma >= x/2.0 or sigma >= y/2.0:
            warn('Warning: sigma is too high for the dimension of the map.')
        if random_seed:
            self.random_generator = random.RandomState(random_seed)
        else:
            self.random_generator = random.RandomState(random_seed)
        if decay_function:
            self._decay_function = decay_function
        else:
            self._decay_function = lambda x, t, max_iter: x/(1+t/max_iter)
        self.learning_rate = learning_rate
        self.sigma = sigma
        self.weights = self.random_generator.rand(x,y,input_len)*2-1 # random initialization
        for i in range(x):
            for j in range(y):
                self.weights[i,j] = self.weights[i,j] / fast_norm(self.weights[i,j]) # normalization
        self.activation_map = zeros((x,y))
        self.neigx = arange(x)
        self.neigy = arange(y) # used to evaluate the neighborhood function
        self.neighborhood = self.gaussian

    def _activate(self, x):
        """ Updates matrix activation_map, in this matrix the element i,j is the response of the neuron i,j to x """
        s = subtract(x, self.weights) # x - w
```

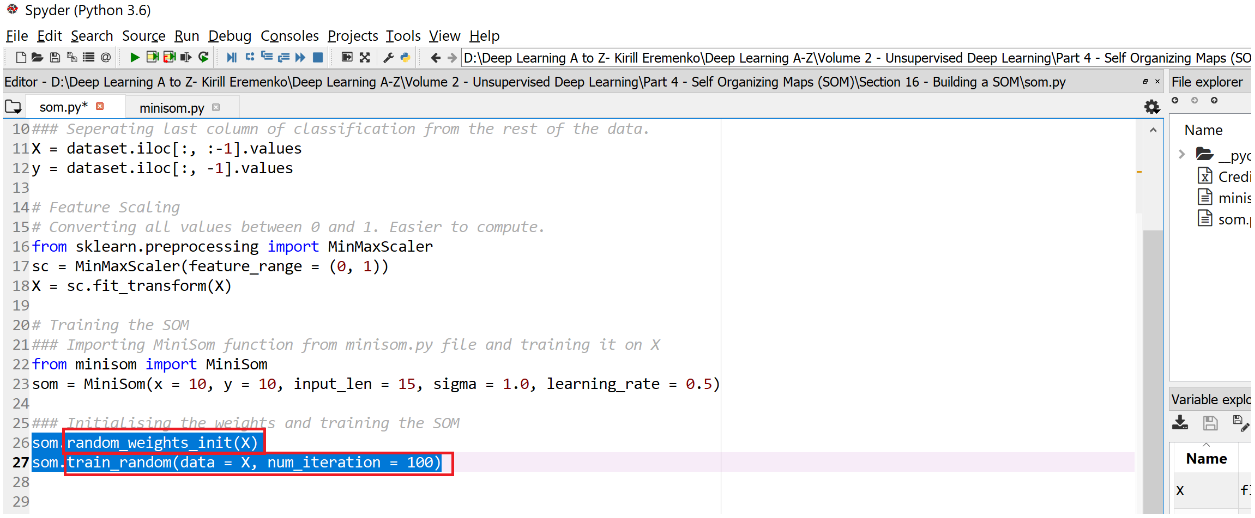
```
Python 3.6.4 |Ana
Type "copyright",

IPython 6.2.1 --

In [1]:
```

We will import the MiniSom class from the python code. The object created from it will need certain parameters. The parameters X and Y are the dimensions of the grid, the grid being the SOM itself. Choosing the larger grid will lead to larger accuracy, but our dataset is not that large, so we will be using a simple grid of 10*10. In this case, the X and Y parameters would be 10 each. The next parameter is input_length, which is the number of features in our dataset X (which is 15). Sigma is the radius of neighborhood, which we will keep default. The learning_rate is the hyperparameter which will adjust the weights on each iteration, we will keep its default value. The decay_function parameter is used for improving convergence, but we will keep it at default none. random_seed will also be default None.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
Size
__pycache__
Credit_Card_Applications.csv
minisom.py
som.py
Variable explorer

```python
### Seperating last column of classification from the rest of the data.
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Feature Scaling
# Converting all values between 0 and 1. Easier to compute.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

# Training the SOM
### Importing MiniSom function from minisom.py file and training it on X
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0, learning_rate = 0.5)


```

Before training the SOM, we need to initialize its weights with values close but not equal to 0. We can do this using the random_weights_init () function in the minisom implementation, which will initialize the SOM weights. The training can be done using the train_random () function and passing the input X and number of iterations as 100 as parameters.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SO
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
_pyc
Credi
minis
som.
Variable explo
Name
X
f

```python
### Seperating last column of classification from the rest of the data.
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Feature Scaling
# Converting all values between 0 and 1. Easier to compute.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

# Training the SOM
### Importing MiniSom function from minisom.py file and training it on X
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0, learning_rate = 0.5)

### Initialising the weights and training the SOM
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)


```

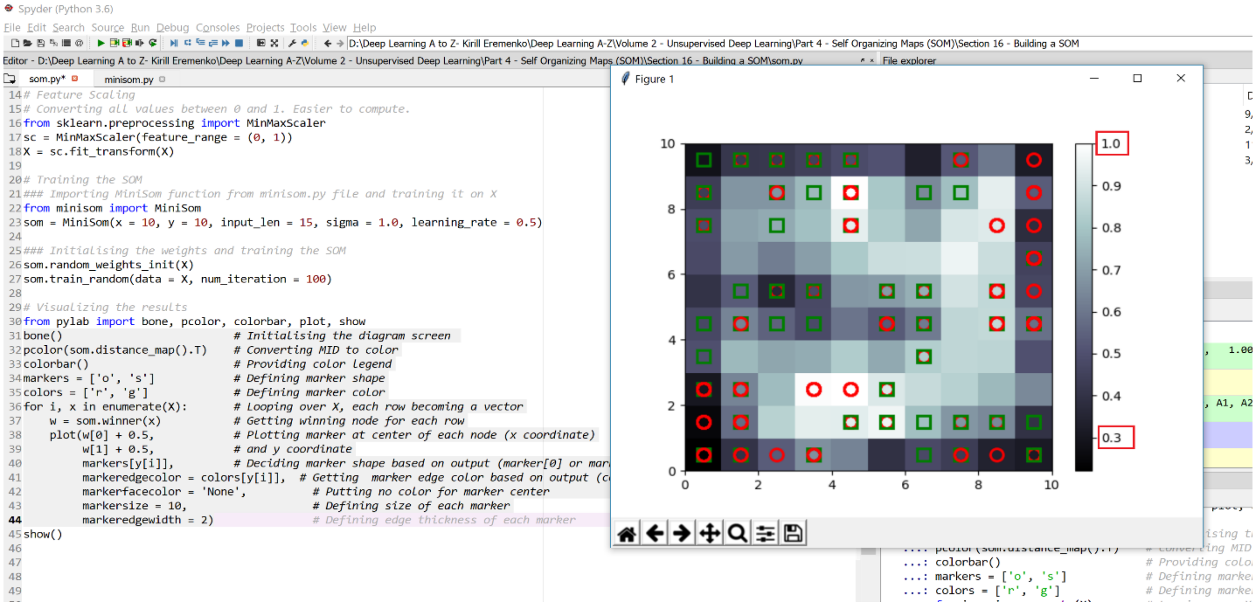
Visualizing the Results

After training, we will plot the self-organizing map itself. We will see a 2d grid that will contain all the "winning nodes". For each of these nodes we will get the MID (Mean Inter-Neuron Distance), which is basically the mean distance of all its neighboring nodes, defined by the radius during our SOM creation. The higher the MID, the "far away" the winning node will be from its neighbors, the more likelihood of it being an outlier.

We will not be seeing numbers, we will use colors. The higher the MID of a winning node, the closer its color will be towards white.

We will not be using pyplot or matplotlib, because we are dealing with a SOM, and not a regular histogram. We will be using the components bone, pcolor, colorbar, plot and show from the pylab library.

We will initialize the window where the visualization will appear using bone () function. We will use the distance_map method to get the MID from the object som. Then we will feed the transpose of this into pcolor () function to convert them into colors. This will give us a basic SOM grid, but we still have not defined whether whiter colors are for higher or lower MIDs or vice-versa. The function colorbar () will provide this legend for us. We can now create markers to focus on customers who cheated and got approved rather than customers who cheated and did not get approval. We will use green squares to mark customers who got approval and red circles to mark those who did not get approval. We will create a vector called markers, containing 'o' for circles and 's' for squares. Similarly, we will have a vector for colors, having 'r' for red and 'g' for green. We will loop over the X and y to get if a customer was approved or not, and to make the correct mark on the right winning node. We use the function winner () to get the winning node of a row (vector).

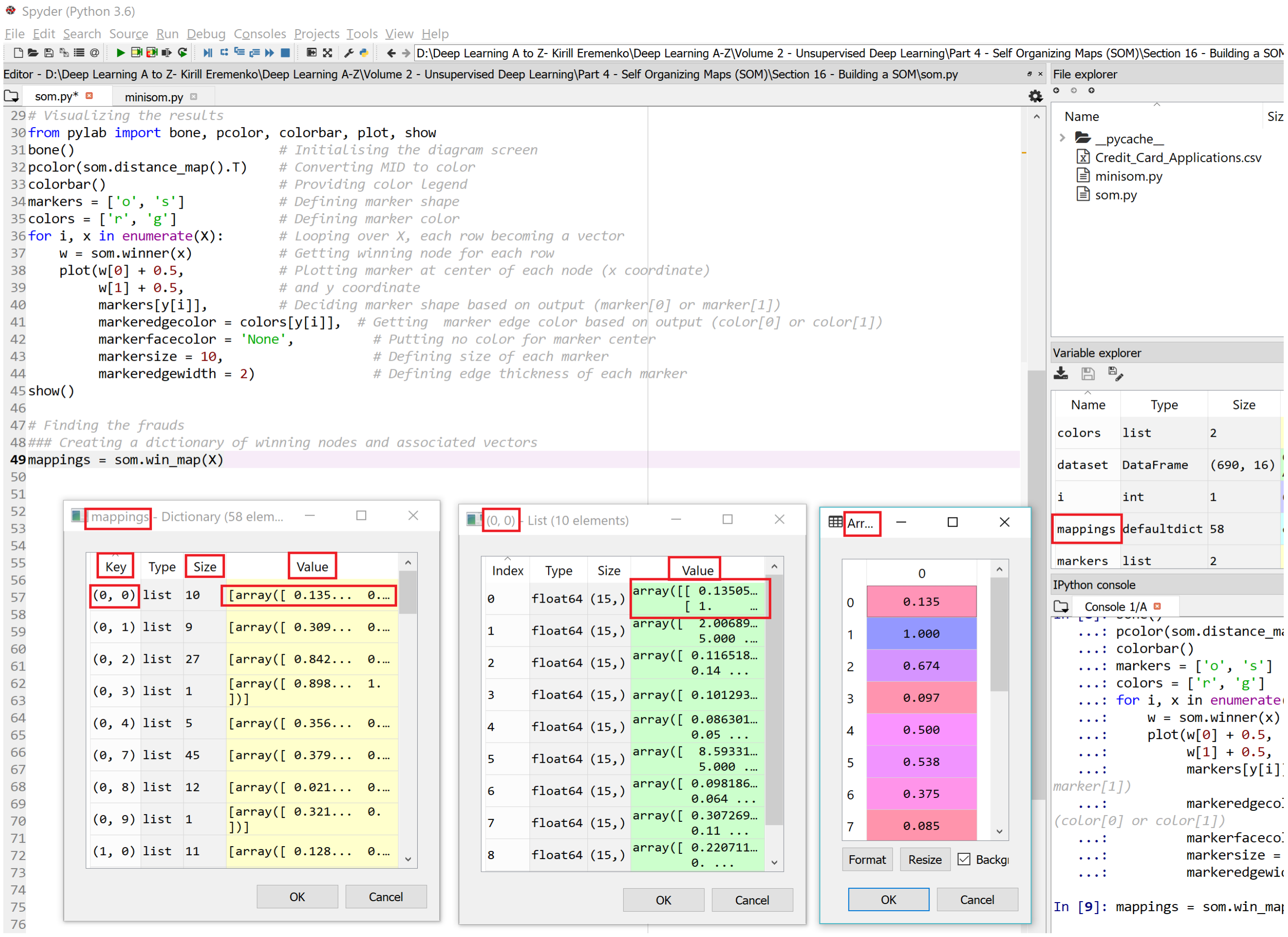

Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
File explorer
som.py*
minisom.py
Figure 1
10
8
6
4
2
0
0
2
4
6
8
10
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3

```
# Feature Scaling
# Converting all values between 0 and 1. Easier to compute.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

# Training the SOM
### Importing MiniSom function from minisom.py file and training it on X
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0, learning_rate = 0.5)

### Initialising the weights and training the SOM
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)

# Visualizing the results
from pylab import bone, pcolor, colorbar, plot, show
bone()                            # Initialising the diagram screen
pcolor(som.distance_map().T)      # Converting MID to color
colorbar()                        # Providing color legend
markers = ['o', 's']              # Defining marker shape
colors = ['r', 'g']               # Defining marker color
for i, x in enumerate(X):         # Looping over X, each row becoming a vector
    w = som.winner(x)             # Getting winning node for each row
    plot(w[0] + 0.5,              # Plotting marker at center of each node (x coordinate)
         w[1] + 0.5,              # and y coordinate
         markers[y[i]],           # Deciding marker shape based on output (marker[0] or mar
         markeredgecolor = colors[y[i]],  # Getting  marker edge color based on output (c
         markerfacecolor = 'None',          # Putting no color for marker center
         markersize = 10,                   # Defining size of each marker
         markeredgewidth = 2)               # Defining edge thickness of each marker
show()
```

We can see the SOM along with the markers. We notice that the outliers contain cases of both approval and rejection as well.

Finding the Frauds

There is no inverse mapping function which will help us catch the frauds. But there is a dictionary implemented in MiniSom called win_map, which will map the winning nodes to their rows i.e. Customers. The Key in the dictionary is the coordinate of the winning node, (0,0) will be lower left winning node. The Size tells us the number of rows associated with the winning node. On clicking the Value field, we get a list of vectors associated with the winning node. On clicking on one of the vectors, we get a list of its attributes, which have been scaled of course.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Variable explorer
IPython console
Console 1/A

```
# Visualizing the results
from pylab import bone, pcolor, colorbar, plot, show
bone()                          # Initialising the diagram screen
pcolor(som.distance_map().T)    # Converting MID to color
colorbar()                      # Providing color legend
markers = ['o', 's']            # Defining marker shape
colors = ['r', 'g']             # Defining marker color
for i, x in enumerate(X):       # Looping over X, each row becoming a vector
    w = som.winner(x)           # Getting winning node for each row
    plot(w[0] + 0.5,            # Plotting marker at center of each node (x coordinate)
         w[1] + 0.5,            # and y coordinate
         markers[y[i]],         # Deciding marker shape based on output (marker[0] or marker[1])
         markeredgecolor = colors[y[i]],  # Getting  marker edge color based on output (color[0] or color[1])
         markerfacecolor = 'None',          # Putting no color for marker center
         markersize = 10,                   # Defining size of each marker
         markeredgewidth = 2)               # Defining edge thickness of each marker
show()

# Finding the frauds
### Creating a dictionary of winning nodes and associated vectors
mappings = som.win_map(X)
```

mappings - Dictionary (58 elem...

Key	Type	Size	Value
(0, 0)	list	10	[array([0.135... 0....
(0, 1)	list	9	[array([0.309... 0....
(0, 2)	list	27	[array([0.842... 0....
(0, 3)	list	1	[array([0.898... 1.])]
(0, 4)	list	5	[array([0.356... 0....
(0, 7)	list	45	[array([0.379... 0....
(0, 8)	list	12	[array([0.021... 0....
(0, 9)	list	1	[array([0.321... 0.])]
(1, 0)	list	11	[array([0.128... 0....

OK Cancel

(0, 0) - List (10 elements)

Index	Type	Size	Value
0	float64	(15,)	array([[0.13505... [1. ...
1	float64	(15,)	array([2.00689... 5.000 ....
2	float64	(15,)	array([0.116518... 0.14 ...
3	float64	(15,)	array([0.101293...
4	float64	(15,)	array([0.086301... 0.05 ...
5	float64	(15,)	array([8.59331... 5.000 ....
6	float64	(15,)	array([0.098186... 0.064 ...
7	float64	(15,)	array([0.307269... 0.11 ...
8	float64	(15,)	array([0.220711... 0. ...

OK Cancel

Arr...

	0
0	0.135
1	1.000
2	0.674
3	0.097
4	0.500
5	0.538
6	0.375
7	0.085

Format Resize ☑ Backgı

OK Cancel

File explorer

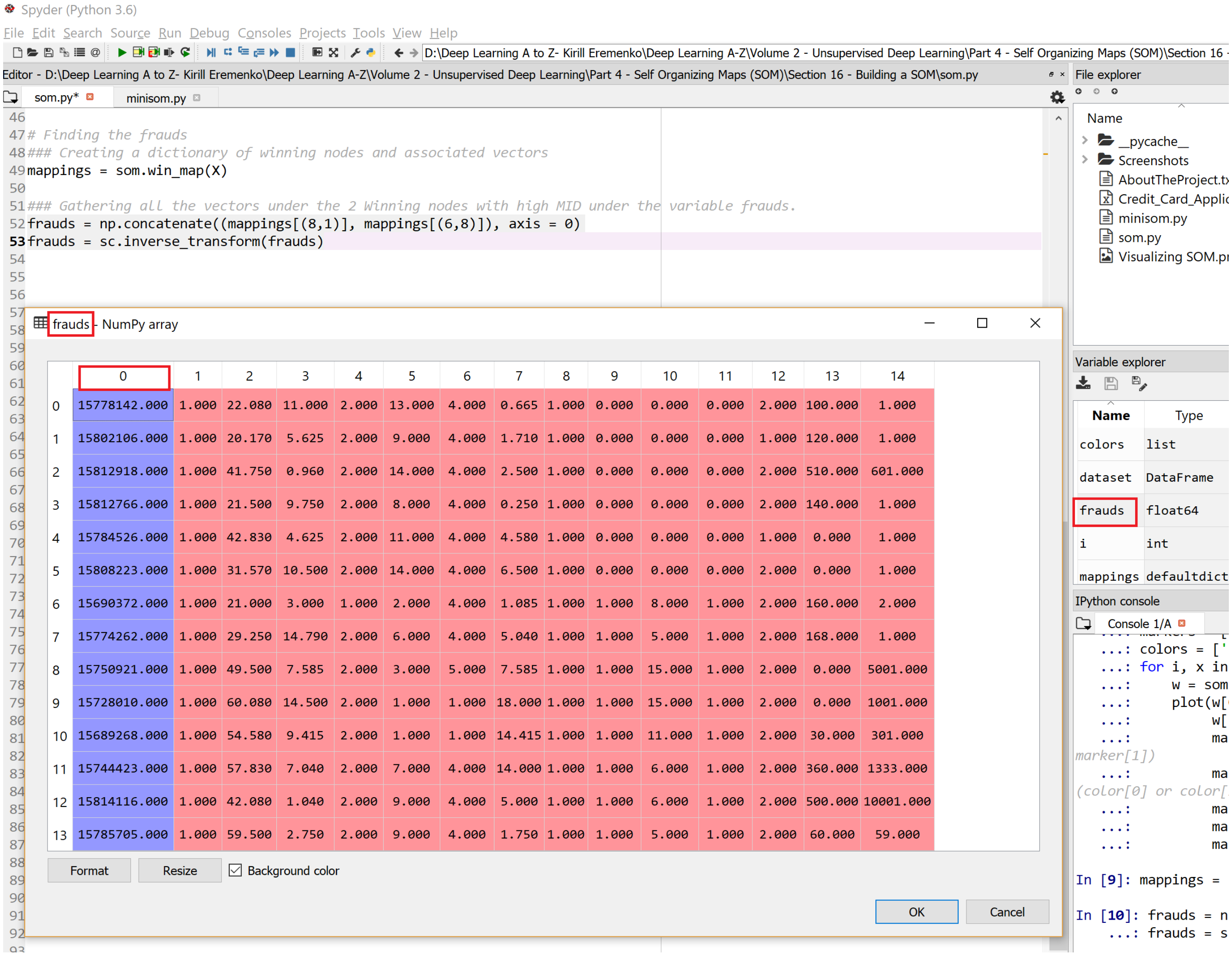
Name	Siz
__pycache__	
Credit_Card_Applications.csv	
minisom.py	
som.py	

Variable explorer

Name	Type	Size
colors	list	2
dataset	DataFrame	(690, 16)
i	int	1
mappings	defaultdict	58
markers	list	2

```
   ...: pcolor(som.distance_ma
   ...: colorbar()
   ...: markers = ['o', 's']
   ...: colors = ['r', 'g']
   ...: for i, x in enumerate
   ...:     w = som.winner(x)
   ...:     plot(w[0] + 0.5,
   ...:          w[1] + 0.5,
   ...:          markers[y[i]
marker[1])
   ...:          markeredgeco
(color[0] or color[1])
   ...:          markerfaceco
   ...:          markersize =
   ...:          markeredgewi

In [9]: mappings = som.win_ma
```

From the ‘mappings’ dictionary, we will collect all the vectors from the two winning nodes with high MID [(8,1) and (6,8)] and concatenate them as a numpy array. We then use the inverse_transform function of MinMaxScaler library to get the original values of the attributes for all the fraud vectors.


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 -
Editor - D:\Deep Learning A to Z- Kirill Eremenko\Deep Learning A-Z\Volume 2 - Unsupervised Deep Learning\Part 4 - Self Organizing Maps (SOM)\Section 16 - Building a SOM\som.py
som.py*
minisom.py
File explorer
Name
__pycache__
Screenshots
AboutTheProject.tx
Credit_Card_Applic
minisom.py
som.py
Visualizing SOM.p
Variable explorer
IPython console
Console 1/A
frauds - NumPy array
Format
Resize
☑ Background color
OK
Cancel

```
# Finding the frauds
### Creating a dictionary of winning nodes and associated vectors
mappings = som.win_map(X)

### Gathering all the vectors under the 2 Winning nodes with high MID under the variable frauds.
frauds = np.concatenate((mappings[(8,1)], mappings[(6,8)]), axis = 0)
frauds = sc.inverse_transform(frauds)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	15778142.000	1.000	22.080	11.000	2.000	13.000	4.000	0.665	1.000	0.000	0.000	0.000	2.000	100.000	1.000
1	15802106.000	1.000	20.170	5.625	2.000	9.000	4.000	1.710	1.000	0.000	0.000	0.000	1.000	120.000	1.000
2	15812918.000	1.000	41.750	0.960	2.000	14.000	4.000	2.500	1.000	0.000	0.000	0.000	2.000	510.000	601.000
3	15812766.000	1.000	21.500	9.750	2.000	8.000	4.000	0.250	1.000	0.000	0.000	0.000	2.000	140.000	1.000
4	15784526.000	1.000	42.830	4.625	2.000	11.000	4.000	4.580	1.000	0.000	0.000	0.000	1.000	0.000	1.000
5	15808223.000	1.000	31.570	10.500	2.000	14.000	4.000	6.500	1.000	0.000	0.000	0.000	2.000	0.000	1.000
6	15690372.000	1.000	21.000	3.000	1.000	2.000	4.000	1.085	1.000	1.000	8.000	1.000	2.000	160.000	2.000
7	15774262.000	1.000	29.250	14.790	2.000	6.000	4.000	5.040	1.000	1.000	5.000	1.000	2.000	168.000	1.000
8	15750921.000	1.000	49.500	7.585	2.000	3.000	5.000	7.585	1.000	1.000	15.000	1.000	2.000	0.000	5001.000
9	15728010.000	1.000	60.080	14.500	2.000	1.000	1.000	18.000	1.000	1.000	15.000	1.000	2.000	0.000	1001.000
10	15689268.000	1.000	54.580	9.415	2.000	1.000	1.000	14.415	1.000	1.000	11.000	1.000	2.000	30.000	301.000
11	15744423.000	1.000	57.830	7.040	2.000	7.000	4.000	14.000	1.000	1.000	6.000	1.000	2.000	360.000	1333.000
12	15814116.000	1.000	42.080	1.040	2.000	9.000	4.000	5.000	1.000	1.000	6.000	1.000	2.000	500.000	10001.000
13	15785705.000	1.000	59.500	2.750	2.000	9.000	4.000	1.750	1.000	1.000	5.000	1.000	2.000	60.000	59.000

Name	Type
colors	list
dataset	DataFrame
frauds	float64
i	int
mappings	defaultdict

```
   ...: colors = ['
   ...: for i, x in
   ...:     w = som
   ...:     plot(w[
   ...:         w[
   ...:         ma
marker[1])
   ...:         ma
(color[0] or color[
   ...:         ma
   ...:         ma
   ...:         ma

In [9]: mappings =

In [10]: frauds = n
    ...: frauds = s
```