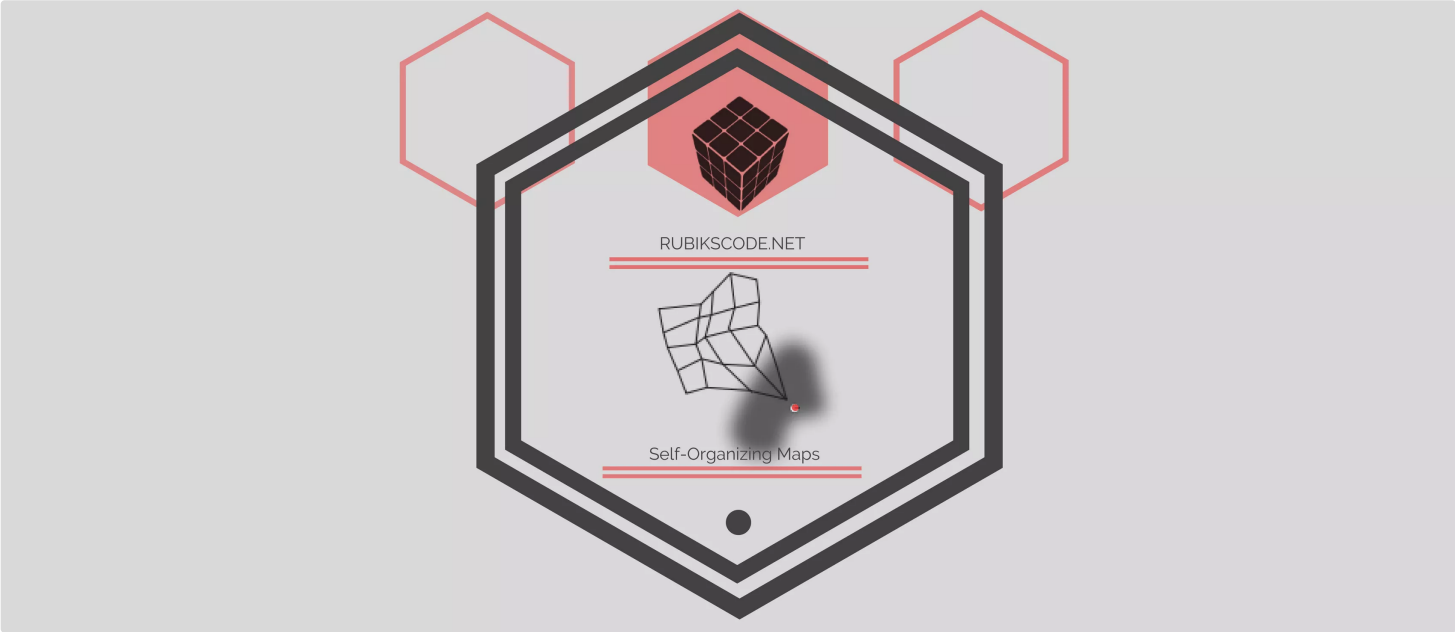
RUBIK'S CODE

Freedom.
Wisdom.
Excellence.

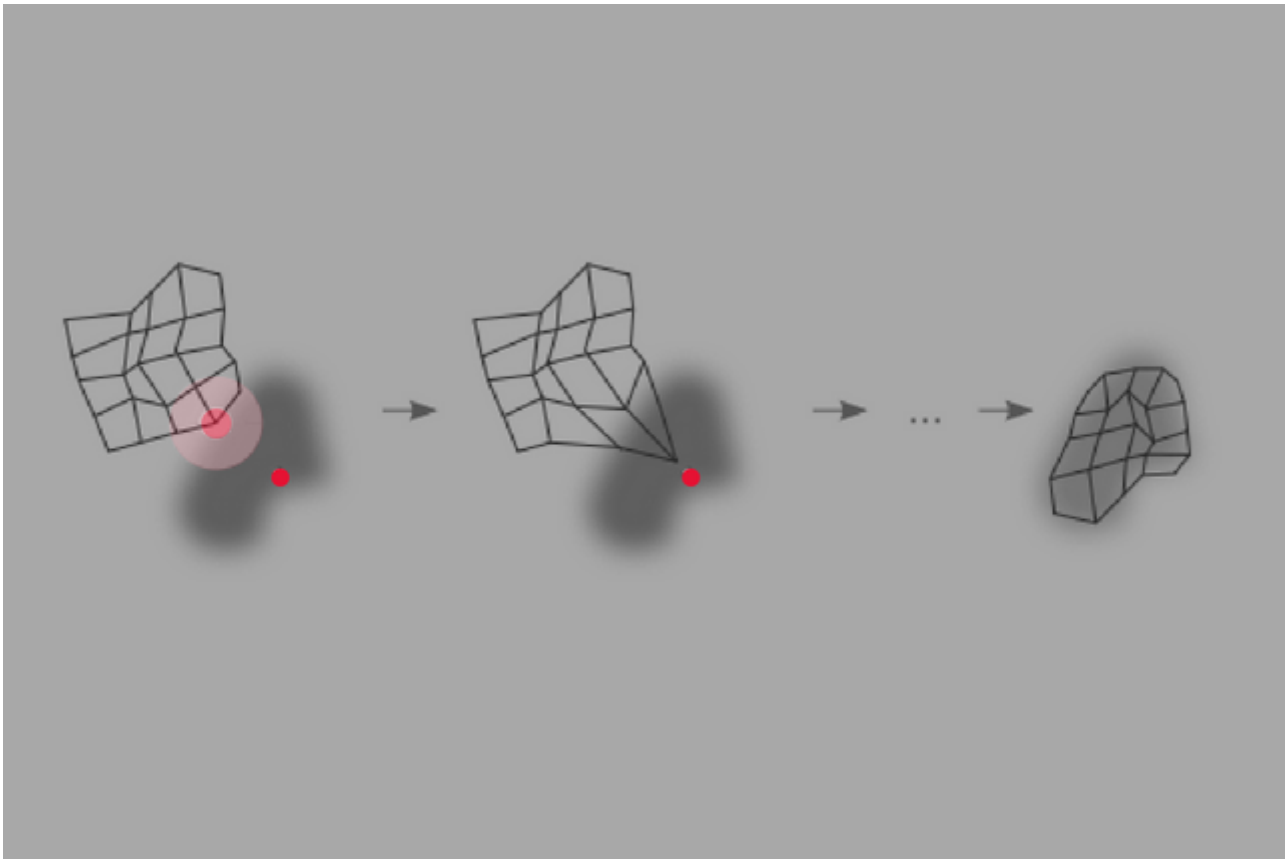
Credit Card Fraud Detection Using Self-Organizing Maps and Python

SEPTEMBER 24, 2018 — LEAVE A COMMENT



In the previous three articles, we explored the world of Self-Organizing Maps. First, we got some theoretical background on the subject. Then in the second article, we saw how we could implement Self-Organizing Maps using TensorFlow. After that, in the third article, we have done the same thing in a different technology and implemented Self-Organizing Maps using C#. In all those articles, we focused on how Self-Organizing Maps utilize unsupervised learning for clustering data. While they are using similar mechanisms as standard, feed-forward neural networks these maps are able to cluster input data into different categories without getting expected results beforehand.

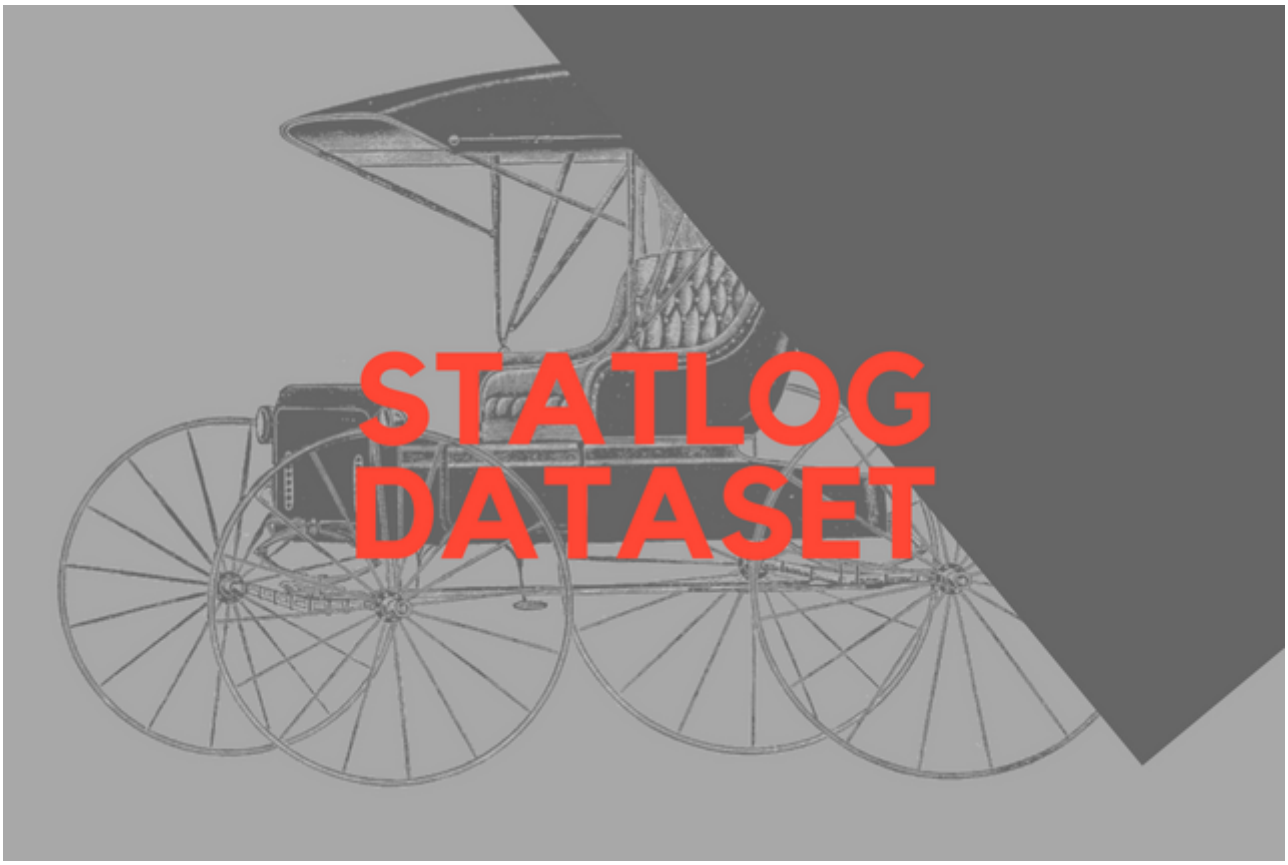
They are able to do so by making a relationship in the input data on their own. It is important to emphasize that concept of neurons, connection and weighted connections are having a different meaning in Self-Organizing Maps. Neurons are grouped into two categories. The first category is a collection of input neurons. Their number corresponds to the number of features that we have in a used dataset. The second category is a collection of output neurons. These neurons are usually organized as one or two-dimensional arrays and are triggered only by certain input values.



It is also important that every neuron in the Self-Organizing map have a location assigned to it. These locations are an important parameter since it is considered that neurons that lie close to each other have similar properties and actually represent a cluster. If you want to learn more details on the structure of Self-Organizing Maps and their learning process, you may do so here.

In this article, we are going to focus more on the ways we can use Self-Organizing Maps in a real-world problem. We will explore how to detect credit card frauds using this mechanism. For that purpose, we will use TensorFlow implementation that we have already made. But before we jump into the solution of the problem, the first thing we need to check out is the dataset we will use for this purpose.

Statlog (Australian Credit Approval) Dataset



In this article, we will use the Statlog (Australian Credit Approval) Dataset. This dataset holds credit card applications. Basically, the bank issues credit cards to customers and they are trying to figure out did any faulty application got approved by mistake. Our goal is to detect possible frauds so they can be further investigated by the bank. This way, the bank can protect itself from possible losses in the future. In the dataset, all attribute names and values have been changed to meaningless symbols to protect the confidentiality of the data. This dataset has a good mix of attributes – continuous, nominal with small numbers of values, and nominal with larger numbers of values, which makes it perfect for the practice. The attributes of this dataset are:

- CustomerID – Id of the customer
- A1 – categorical, possible values: 0 and 1
- A2 – continuous
- A3 – continuous
- A4 – categorical, possible values: 1, 2 and 3
- A5 – categorical, possible values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
- A6 – categorical, possible values: 1, 2, 3, 4, 5, 6, 7, 8, 9
- A7 – continuous
- A8 – categorical, possible values: 0 and 1
- A9 – categorical, possible values: 0 and 1
- A10 – continuous

- A11 – categorical, possible values: 0 and 1
- A12 – categorical, possible values: 1, 2 and 3
- A13 – continuous
- A14 – continuous
- A15 (Class) – class attribute, possible values: 1 and 2

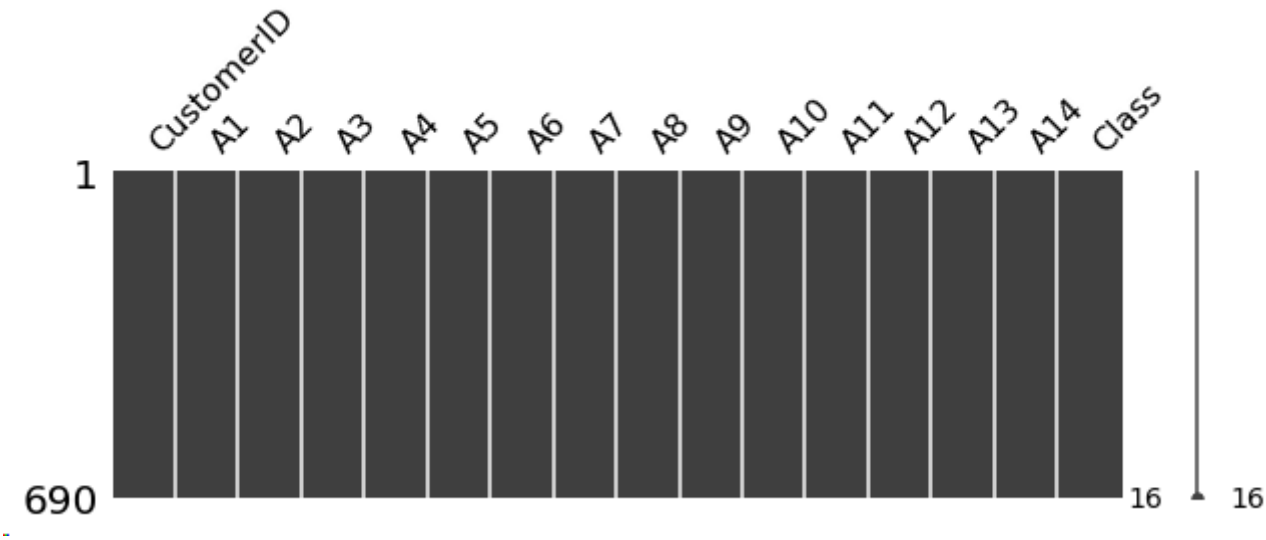
As you can see that features have no semantics, except the final feature which is indicating had credit card been issued to the customer or not. What we want to do is explore attributes from A1 to A14, and use Self-Organizing Map to figure out which customer committed fraud. We will use Python 3.6.5 and Spyder IDE. Also, TensorFlow 1.10.0 version is used. Here you can find a quick guide on how to install it and how to start working with it.

Feature Analysis

Before we proceed with the implementation, let's explore the dataset a little bit. Since we are not having the semantics of each column in the dataset it is important for us to find out as much as we can about the data. Here is what we will get once we load data into a variable:

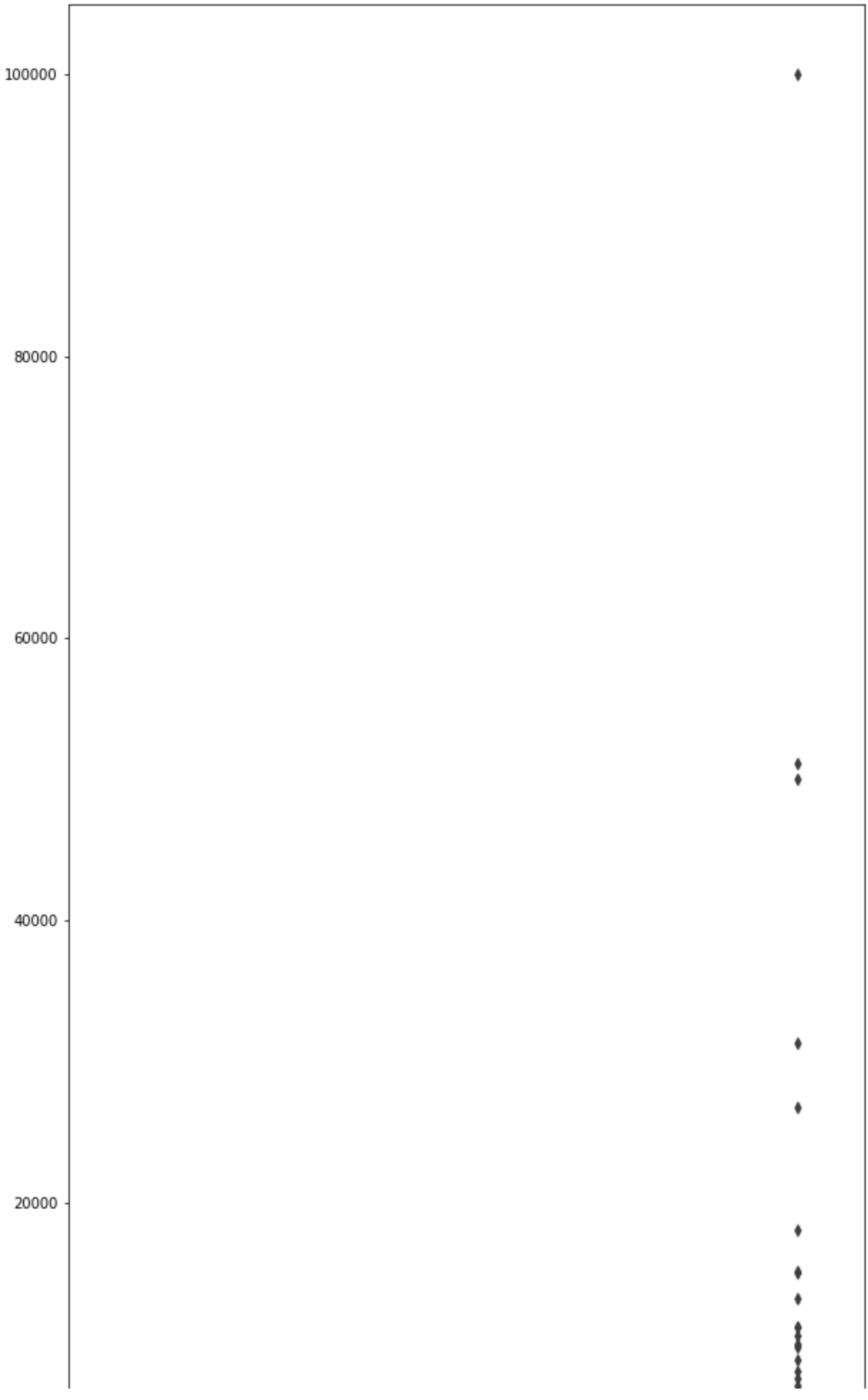
Index	CustomerID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	Class
0	15776156	1	22.08	11.46	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	15739548	0	22.67	7	2	8	4	0.165	0	0	0	0	2	160	1	0
2	15662854	0	29.58	1.75	1	4	4	1.25	0	0	0	1	2	280	1	0
3	15687688	0	21.67	11.5	1	5	3	0	1	1	11	1	2	0	1	1
4	15715750	1	20.17	8.17	2	6	4	1.96	1	1	14	0	2	60	159	1
5	15571121	0	15.83	0.585	2	8	8	1.5	1	1	2	0	2	100	1	1
6	15726466	1	17.42	6.5	2	3	4	0.125	0	0	0	0	2	60	101	0
7	15660390	0	58.67	4.46	2	11	8	3.04	1	1	6	0	2	43	561	1
8	15663942	1	27.83	1	1	2	8	3	0	0	0	0	2	176	538	0
9	15638610	0	55.75	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	41.42	5	2	11	8	5	1	1	6	1	2	470	1	1
12	15609356	1	20.67	1.25	1	8	8	1.375	1	1	3	1	2	140	211	0
13	15803378	1	34.92	5	2	14	8	7.5	1	1	6	1	2	0	1001	1
14	15599440	1	58.58	2.71	2	8	4	2.415	0	0	0	1	2	320	1	0
15	15692408	1	48.08	6.04	2	4	4	0.04	0	0	0	0	2	0	2691	1
16	15683168	1	29.58	4.5	2	9	4	7.5	1	1	2	1	2	330	1	1
17	15790254	0	18.92	9	2	6	4	0.75	1	1	2	0	2	88	592	1
18	15767729	1	20	1.25	1	4	4	0.125	0	0	0	0	2	140	5	0
19	15768600	0	22.42	5.665	2	11	4	2.585	1	1	7	0	2	129	3258	1
20	15699839	0	28.17	0.585	2	6	4	0.04	0	0	0	0	2	260	1005	0
21	15786237	0	19.17	0.585	1	6	4	0.585	1	0	0	1	2	160	1	0
22	15694530	1	41.17	1.335	2	2	4	0.165	0	0	0	0	2	168	1	0
23	15796813	1	41.58	1.75	2	4	4	0.21	1	0	0	0	2	160	1	0
24	15605791	1	19.5	9.585	2	6	4	0.79	0	0	0	0	2	80	351	0
25	15714087	1	32.75	1.5	2	13	8	5.5	1	1	3	1	2	0	1	1
26	15711446	1	22.5	0.125	1	4	4	0.125	0	0	0	0	2	200	71	0

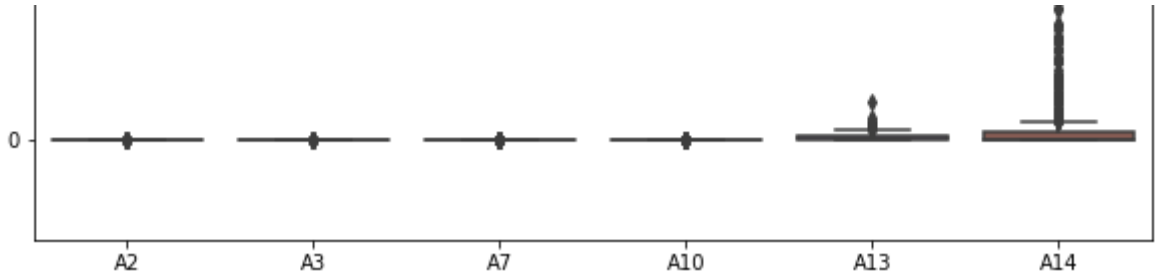
We can see that the first column represents the identification number of the customer. This should be excluded from our calculations since it is not carrying important information. Another thing we can notice from the first glance is that our continuous variables are not in scale, but we will explore that in more details during the outliers detection phase. Let's see if we are missing any data in this dataset. *Missingno* Python library is a great tool for that. Here is what we get if we apply it to our dataset:



We can see that there is no missing data in our dataset, which is a big relief. There would be indicators in *missingno* output if there were some. If that was the case, we would have to make multiple experiments to determine which algorithm should we apply when replacing missing data. This would have to be done because the semantics of the features is unknown.

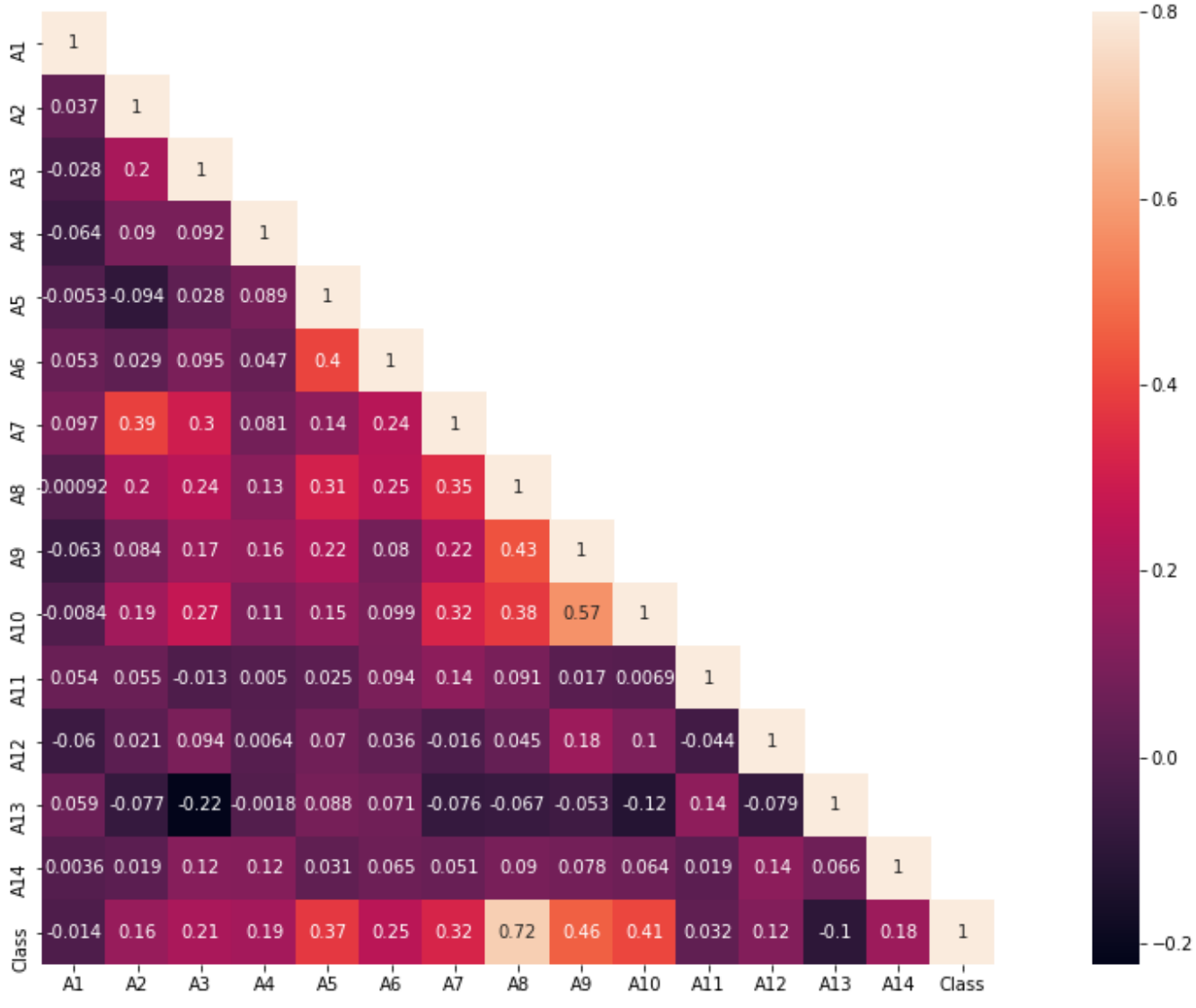
Now, we can proceed with outlier detection. It makes no sense to include categorical data in this analysis, so we extracted only continuous features. Detecting outliers, meaning data samples which are vastly different from the rest of the samples, is actually our main goal and it will solve our whole problem. We are actually trying to detect customers which are standing out of the crowd, so to say. Their unified result should be different from the rest. That is why detecting outliers on every individual feature are important as well. We use *seaborn* library for visualization and we get really interesting results:


100000
80000
60000
40000
20000



We see that majority of our data is out of scale, meaning we will have to apply some sort of scaling algorithm before applying any machine learning algorithm (in this post we will use Self-Organizing Maps, but we could use some other clustering algorithm as well). Also, we see that *A14* is having huge fluctuations, which could be some sort of indication. If we were having some other problem and using some other dataset, we might want to choose to remove outliers from our calculations, but as we mentioned previously, these values are crucial for this problem.

Finally, let's check the correlation between the data in our dataset:



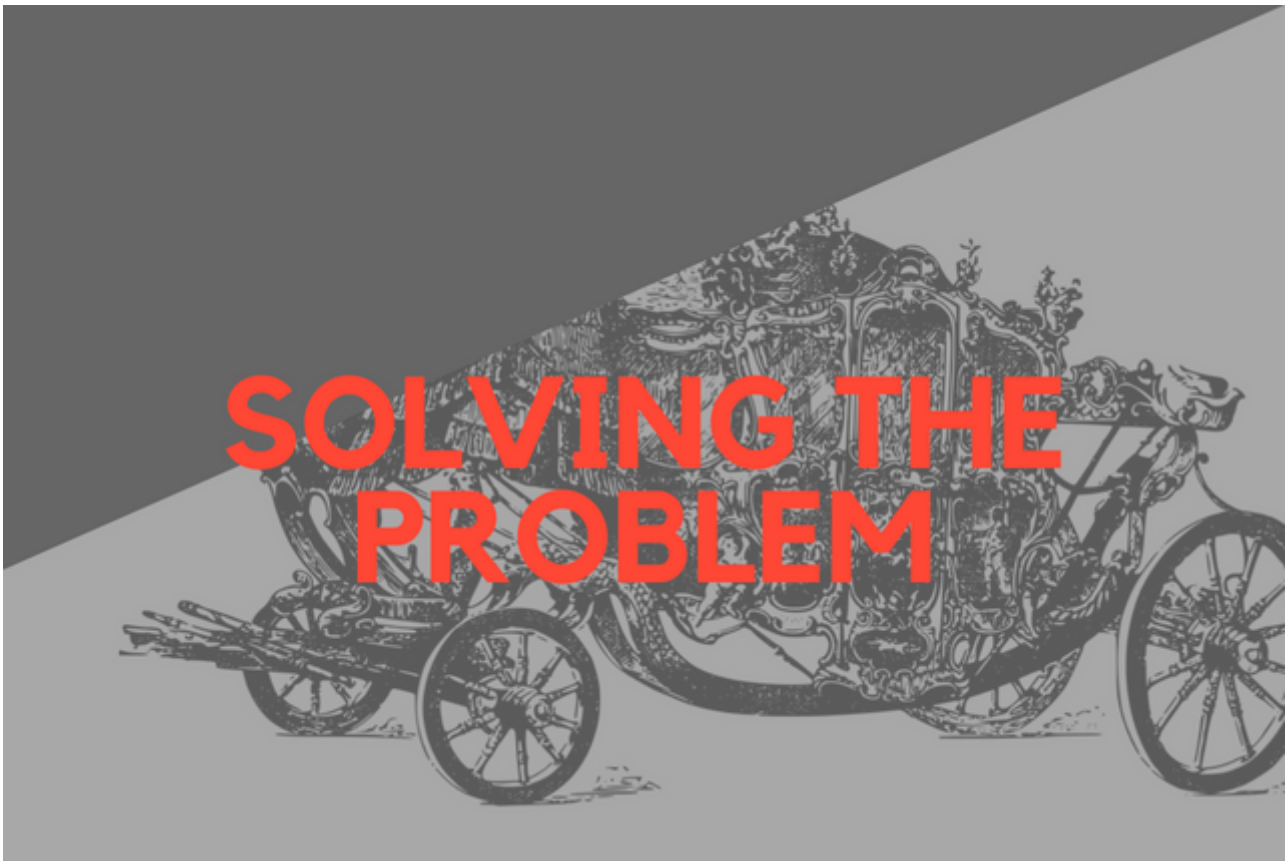
We can see that there is a high correlation value between A8 feature and the final outcome. However, we won't remove this feature at the moment, like we might do in some other problems. It is important to remember this because we could optimize our solution later and get even more precise results.

Here is code that we used for this analysis:

```
import numpy as np
import pandas as pd
import missingno as msno
import seaborn as sn
import matplotlib.pyplot as plt

data = pd.read_csv('Credit_Card_Applications.csv')
data = data.drop(["CustomerID"],axis=1)

# Missing data detection
msno.matrix(data,figsize=(10,3))

# Outliners detection and class imbalance
continiousData = pd.DataFrame()
continousVariableList  = ["A2", "A3", "A7", "A10", "A13", "A14"]
for var in continousVariableList:
    continiousData[var] = data[var].astype("float32")

fig, axes = plt.subplots(nrows=1,ncols=1)
fig.set_size_inches(10, 20)
sn.boxplot(data=continiousData,orient="v",ax=axes)

# Correlation analysis
corrMatt = data.corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corrMatt, mask=mask,vmax=.8, square=True,annot=True)
```

feature_analysis.py hosted with ❤ by GitHub view raw

Solving the Problem



For solving this problem, we will use TensorFlow implementation from one of the previous articles. We are going to do it like this:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv('Credit_Card_Applications.csv')

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

from somtf import SOM
som = SOM(x = 10, y = 10, input_dim=15, learning_rate=0.5, num_iter = 100, radius = 1.0)
som.train(X);
```

somtf_fraud_detection.py hosted with ❤ by GitHub view raw

As you can see, the first thing we have done is loading data into *data* variable. After that, we divided input data from output data. We will only use input data for training the Self-Organizing Map. Here is data after separation:

During the feature analysis, we determined that we will need to scale data. For that purpose, we use *MinMaxScaler* from the Sci-Kit Learn library. This class will scale each individual feature within the defined range. Defined values, in this case, are 0 and 1. This means that each feature will be scaled within 0 and 1. This is how the input looks like after scaling is applied:

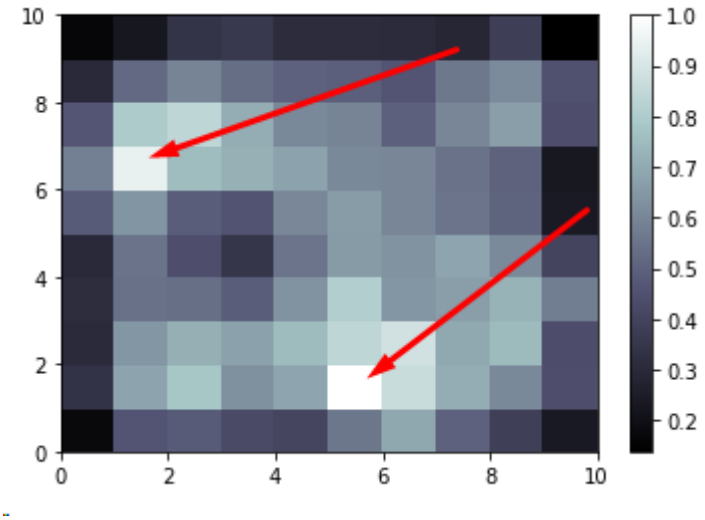
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.842681	1	0.125263	0.409286	0.5	0.230769	0.375	0.055614	0	0	0	1	0.5	0.05	0.01212
1	0.696091	0	0.134135	0.25	0.5	0.538462	0.375	0.00578947	0	0	0	0	0.5	0.08	0
2	0.388982	0	0.238045	0.0625	0	0.230769	0.375	0.0438596	0	0	0	1	0.5	0.14	0
3	0.488425	0	0.119098	0.410714	0	0.307692	0.25	0	1	1	0.164179	1	0.5	0	0
4	0.600795	1	0.0965414	0.291786	0.5	0.384615	0.375	0.0687719	1	1	0.208955	0	0.5	0.03	0.00158
5	0.0216515	0	0.0312782	0.0208929	0.5	0.538462	0.875	0.0526316	1	1	0.0298507	0	0.5	0.05	0
6	0.643706	1	0.055188	0.232143	0.5	0.153846	0.375	0.00438596	0	0	0	0	0.5	0.03	0.001
7	0.379115	0	0.675489	0.159286	0.5	0.769231	0.875	0.106667	1	1	0.0895522	0	0.5	0.0215	0.0056
8	0.393338	1	0.211729	0.0357143	0	0.0769231	0.875	0.105263	0	0	0	0	0.5	0.088	0.00537
9	0.2919	0	0.631579	0.252857	0.5	0.230769	0.875	0.236842	1	1	0.0447761	1	0.5	0.05	0.0005
10	0.31527	1	0.296992	0.0625	0.5	1	0.875	0.157895	1	1	0.0597015	1	0.5	0.1265	0.00857
11	0.0807996	1	0.41609	0.178571	0.5	0.769231	0.875	0.175439	1	1	0.0895522	1	0.5	0.235	0
12	0.174757	1	0.10406	0.0446429	0	0.538462	0.875	0.0482456	1	1	0.0447761	1	0.5	0.07	0.0021
13	0.951688	1	0.318346	0.178571	0.5	1	0.875	0.263158	1	1	0.0895522	1	0.5	0	0.01
14	0.13505	1	0.674135	0.0967857	0.5	0.538462	0.375	0.0847368	0	0	0	1	0.5	0.16	0
15	0.507326	1	0.516241	0.215714	0.5	0.230769	0.375	0.00140351	0	0	0	0	0.5	0	0.0269
16	0.470326	1	0.238045	0.160714	0.5	0.615385	0.375	0.263158	1	1	0.0298507	1	0.5	0.165	0
17	0.899135	0	0.0777444	0.321429	0.5	0.384615	0.375	0.0263158	1	1	0.0298507	0	0.5	0.044	0.00591
18	0.808937	1	0.093985	0.0446429	0	0.230769	0.375	0.00438596	0	0	0	0	0.5	0.07	4e-05
19	0.812425	0	0.130376	0.202321	0.5	0.769231	0.375	0.0907018	1	1	0.104478	0	0.5	0.0645	0.03257
20	0.537082	0	0.216842	0.0208929	0.5	0.384615	0.375	0.00140351	0	0	0	0	0.5	0.13	0.01004
21	0.883049	0	0.0815038	0.0208929	0	0.384615	0.375	0.0205263	1	0	0	1	0.5	0.08	0
22	0.515823	1	0.412331	0.0476786	0.5	0.0769231	0.375	0.00578947	0	0	0	0	0.5	0.084	0

Finally, we will use TensorFlow implementation and train Self-Organizing Map. We are using 10×10 map for representing this data. Start learning rate is 0.5, the initial radius is 1.0 and training is done in 100 iterations.

Once training is complete, we want to get MID, or Mean Inter-neuron Distances between neurons. If you remember, we are trying to detect outliers, meaning the results that stand out from the others. Now, we are having 10×10 map of neurons, which represent clustered

customers by their features. Each neuron has a certain value. If we calculate differences between those values we get Mean Inter-neuron Distances. This distance tells us which neuron is more different than the other and with that which customers are standing out, ie. which ones committed fraud.

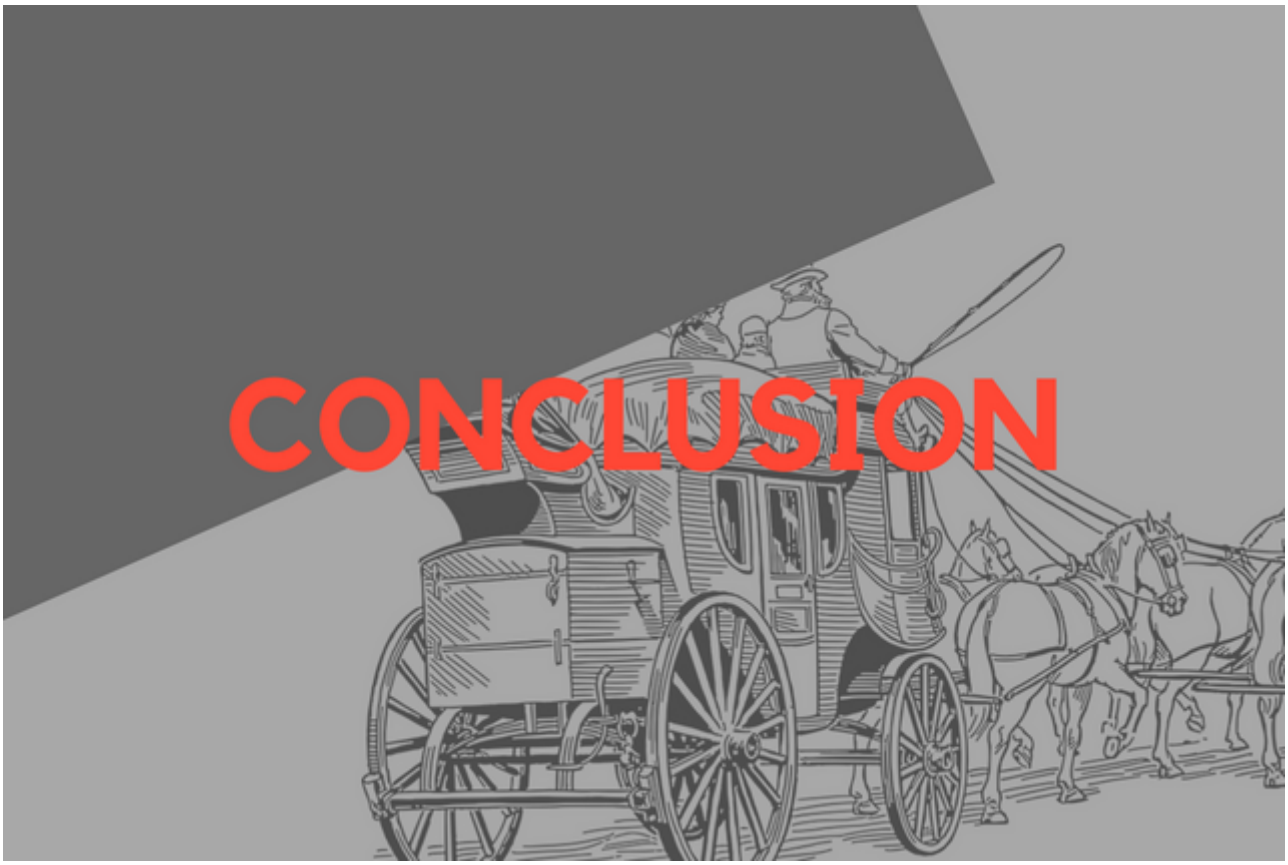
If we map out those distances we will get something like this:



So in our 10×10 map of nodes, the ones with the coordinates (1, 6) and (5, 1) deviate the most. This means that those customers which are connected to these neurons are more likely to commit the fraud than the rest of them. So, when we reverse map those we get the list of potentials frauds and there are 23 of them:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.56519e+07	0	38.75	1.5	2	1	1	0	0	0	0	0	2	76	1
1	1.57161e+07	0	22.25	1.25	1	1	1	3.25	0	0	0	0	2	280	1
2	1.56515e+07	0	69.5	6	2	1	1	0	0	0	0	0	1	0	1
3	1.56726e+07	0	36.75	4.71	2	1	1	0	0	0	0	0	2	160	1
4	1.57523e+07	0	21.08	5	1	1	1	0	0	0	0	0	2	0	1
5	1.57365e+07	0	31.57	11.25	2	1	1	0	0	0	0	0	2	184	5201
6	1.57792e+07	0	27.33	1.665	2	1	1	0	0	0	0	0	2	340	2
7	1.56114e+07	0	23.58	0.585	1	1	1	0.125	0	0	0	0	2	120	88
8	1.56365e+07	0	52.17	0	1	1	1	0	0	0	0	0	2	0	1
9	1.561e+07	0	20.42	10.5	1	14	8	0	0	0	0	1	2	154	33
10	1.55682e+07	0	19.5	0.165	2	11	4	0.04	0	0	0	1	2	380	1
11	1.56622e+07	0	29.75	0.665	2	9	4	0.25	0	0	0	1	2	300	1
12	1.57346e+07	0	21.75	11.75	2	8	4	0.25	0	0	0	1	2	180	1
13	1.55705e+07	0	51.92	6.5	2	3	5	3.085	0	0	0	1	2	73	1
14	1.56417e+07	0	50.25	0.835	2	6	4	0.5	0	0	0	1	2	240	118
15	1.56601e+07	0	26.17	2	2	5	3	0	0	0	0	1	2	276	2
16	1.56036e+07	0	22.92	1.25	2	11	4	0.25	0	0	0	1	2	120	810
17	1.56647e+07	0	24.83	4.5	2	9	4	1	0	0	0	1	2	360	7
18	1.56458e+07	0	18.08	0.375	3	13	1	10	0	0	0	1	1	300	1
19	1.55679e+07	0	23.5	1.5	2	9	4	0.875	0	0	0	1	2	160	1
20	1.56388e+07	0	27.67	2.04	2	9	4	0.25	0	0	0	1	2	180	51
21	1.5641e+07	0	25.25	12.5	2	2	4	1	0	0	0	1	2	180	1063
22	1.56066e+07	0	32.25	1.5	2	8	4	0.25	0	0	0	1	2	372	123

Conclusion



In previous articles, we talked a lot about the possibilities of Self-Organizing Maps, while in this article we utilized them and saw how one can use this type of neural networks in a real-world example. We also saw how we can find our way around dataset even though we don't know semantic behind the features, and that unsupervised learning can connect the dots on its own. To sum it up, we used all the nice things we learned in previous articles and used it in a practical example.

Thank you for reading!

This article is a part of Artificial Neural Networks Series, which you can check out **here**.

Read more posts from the author at **Rubik's Code**.
