ANALYSIS OF COVID-19 WITH SELF ORGANIZING MAPS

A case study on COVID-19

Abstract

The Self-Organizing Map is one of the most popular neural network models. The Self-Organizing Map defines an ordered mapping, a kind of projection from a set of given data items onto a regular, usually two-dimensional grid. Coronavirus disease 2019 (COVID-19) is an infectious disease caused by severe acute respiratory syndrome.

As a case study that shows how to use self organizing maps to compare countries in pandemic situations.Based on the great no of features(eg. Total cases, population, test per million, daily cases, etc.) the SOM illustrates rather refined relationships between the countries two-dimensionally. This method has various applications.

Gajendra Singh
gajendrasingh@jklu.edu.in

Department of Computer Science and Engineering
Institute of Engineering
JK Lakshmipat University, Jaipur, India

1. Case Synopsis

The Self-Organizing Map is one of the most popular neural network models. The Self-Organizing Map defines an ordered mapping, a kind of projection from a set of given data items onto a regular, usually two-dimensional grid.

Coronavirus disease 2019 (**COVID-19**) is an infectious disease caused by severe acute respiratory syndrome.

As a case study that shows how to use self organizing maps to compare countries in pandemic situations.Based on the great no of features(eg. Total cases, population, test per million, daily cases, etc.) the SOM illustrates rather refined relationships between the countries two-dimensionally. This method has various applications.

This case study is done on the Data last updated on 15 April 2020.

This case study represents mapping of various countries on the basis of covid-19 data set. This is done by making a machine learning model by using self organizing maps. Furthermore this map will provide the insights about the global pandemic situation.

2. Learning Objectives

The primary objective of the case is to demonstrate how Self Organizing maps can be used to represent multidimensional data in much lower dimensional spaces - usually one or two dimensions.

Other learning objectives include the following :

1. Demonstrate how a competitive learning approach is used in clustering Countries in Global Pandemic situations.

2. Understanding Rescaling of data using Standard Normalization.

3. Develop a clustering Model using Artificial Neural Networks such as Self Organizing Maps.

4. Evaluate the Model using various measures like Quantization error.

3. Study Questions

1. Develop a clustering model and map the given Countries on the basis of similarity in the situations of covid-19 .

2. Compare the model before normalizing the data and after normalizing the data.

3. Compare the feature plane for all the features.

4. Identify the most relevant feature planes using visualizations.

4. Suggested Answers of Study Questions

4.1 Develop a clustering model and map the given Countries on the basis of similarity in the situations of covid-19 .

The primary objective of this case question is to learn model building in clustering using self organizing maps. The first step in building the model is to understand the data. The data contains mostly continuous variables which is good otherwise we would have to convert them to continuous variable.

Before starting making the model the data should be scaled from 0 to 1 to make model work efficiently. This is done using pre-processing package of sklearn library in python language.

For making model we are using Minisom library which is the implementation of Self Organizing Maps in Python.

Now we have plotted the Map using matplotlib library in python language.

4.2 Compare the model before normalizing the data and after normalizing the data.

Now, we have trained the model two times one before normalizing the data(on raw data) and one after normalizing the data.

It is observed that after normalization of data the quantization error has decreased drastically.(i.e before normalization of data it was 0.3622 and after normalization it was 0.0364)

It has also been observed that dark regions in the Map are more effected by covid-19 in present.

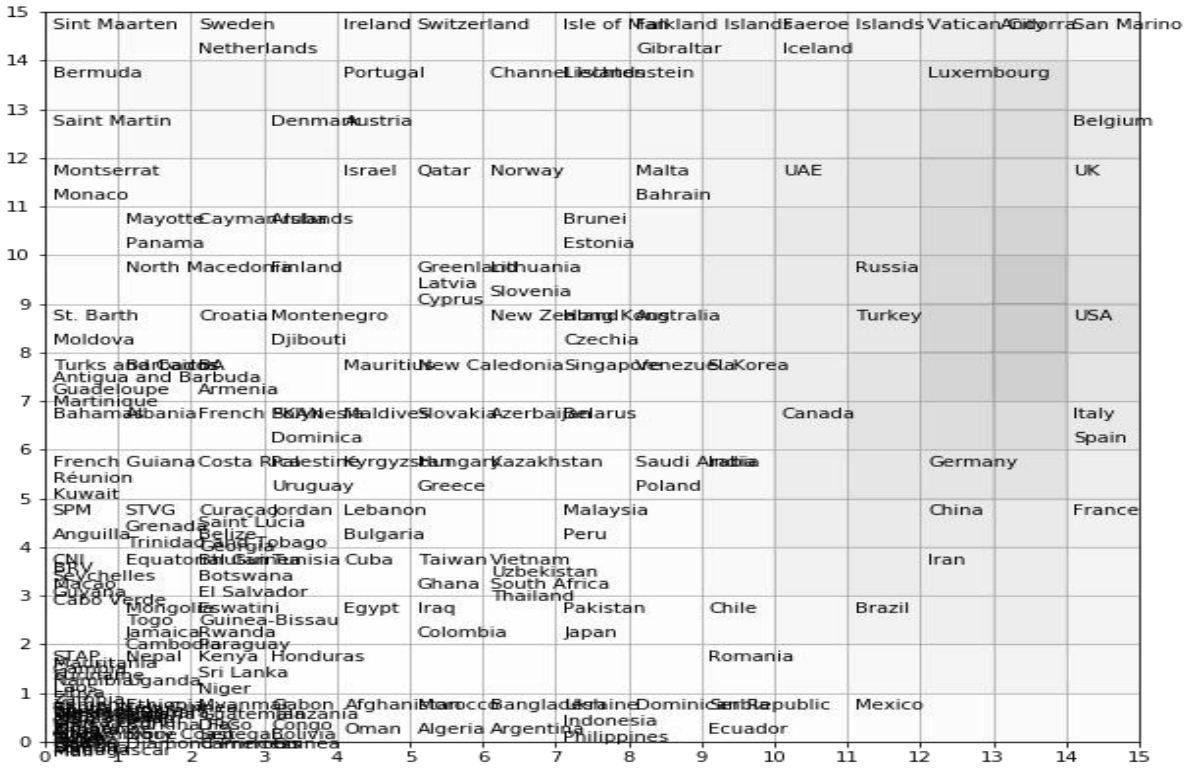


Figure: Map generated before normalizing the data

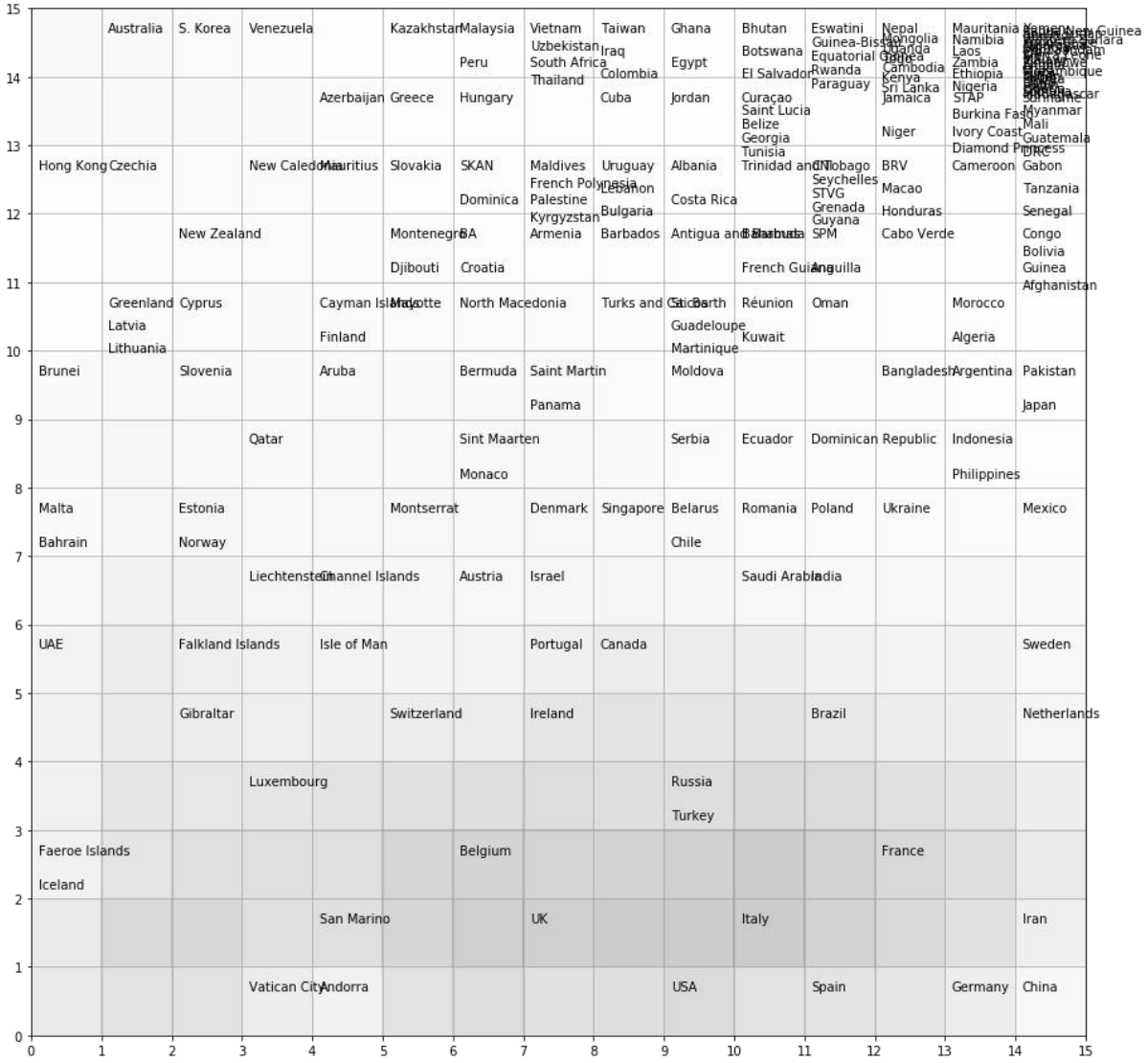


Figure : Map generated after normalization of data

4.3 Compare the feature plane for all the features.

After training the model we have now plotted maps for individual features to draw some insights.

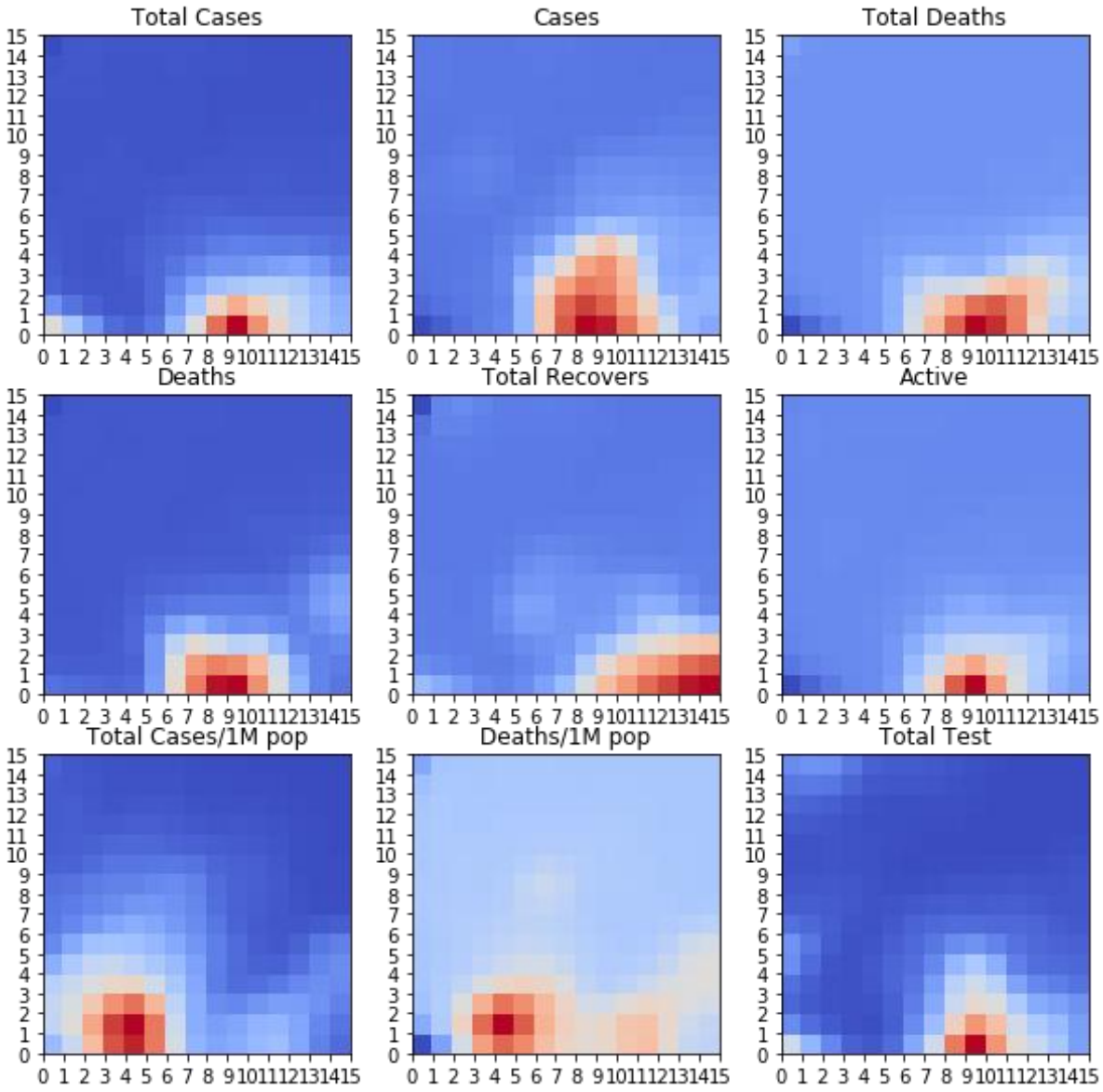


Figure : feature planes for individual features.

In the above map the red area indicates the most affected countries of that particular feature.

It is clearly observed that there are more countries that have higher daily cases then the countries that have higher total cases.

It is also observed that the the countries that have higher no of recovered are not the countries that have higher total cases or higher daily cases.

By the above maps it is clearly indicated that total deaths and daily deaths are correlated with the daily cases. Similarly total cases, active cases, and total test conducted are also correlated.

4.4 Identify the most relevant feature planes using visualizations.

Now we have plotted the most relevant feature plane using som's weights and matplotlib library.

Figure : Most relevant feature plane.

From the above visualization it is clearly observed that our model is mostly affected by Tests conducted per million population, Total cases per million population, daily cases, and Total recovered.

It is least affected by the Total deaths and Total test conducted.

5. References

1) The Self-Organizing Map by TEUVO KOHONEN, SENIOR MEMBER IEEE

Link : https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1990-Kohonen-PIEEE.pdf

2) Honkela, Timo. Description of Kohonen's Self-Organizing Map.

Link : http://www.mlab.uiah.fi/~timo/som/thesis-som.html

3) Kohonen, Teuvu et al. *Self Organization of a Massive Document Collection*. Helsinki University of Technology, 2000

4) Zheng, Gaolin. *Self-Organizing Map (SOM) and Support Vector Machine (SVM)*.

Link : http://www.msci.memphis.edu/~giri/compbio/f00/GZheng.htm

Appendix A

The Self-Organizing Map is one of the most popular *artificial neural network* models. It belongs to the category of competitive learning networks. The Self-Organizing Map is based on unsupervised learning, which means that no human intervention is needed during the learning and that little needs to be known about the characteristics of the input data.

1. Artificial Neural Networks:

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Every node in one layer is connected to every other node in the next layer. We make the network deeper by increasing the number of hidden layers.

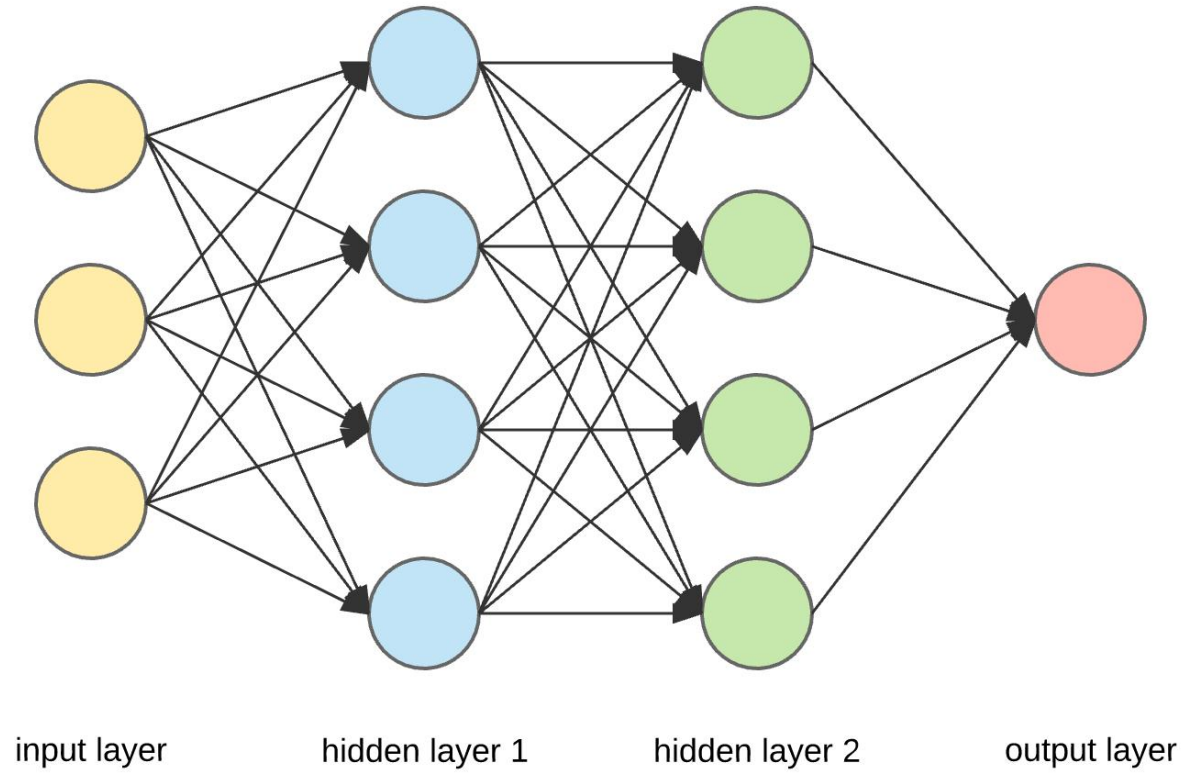


Figure 1

If we zoom in to one of the hidden or output nodes, what we will encounter is the figure below.

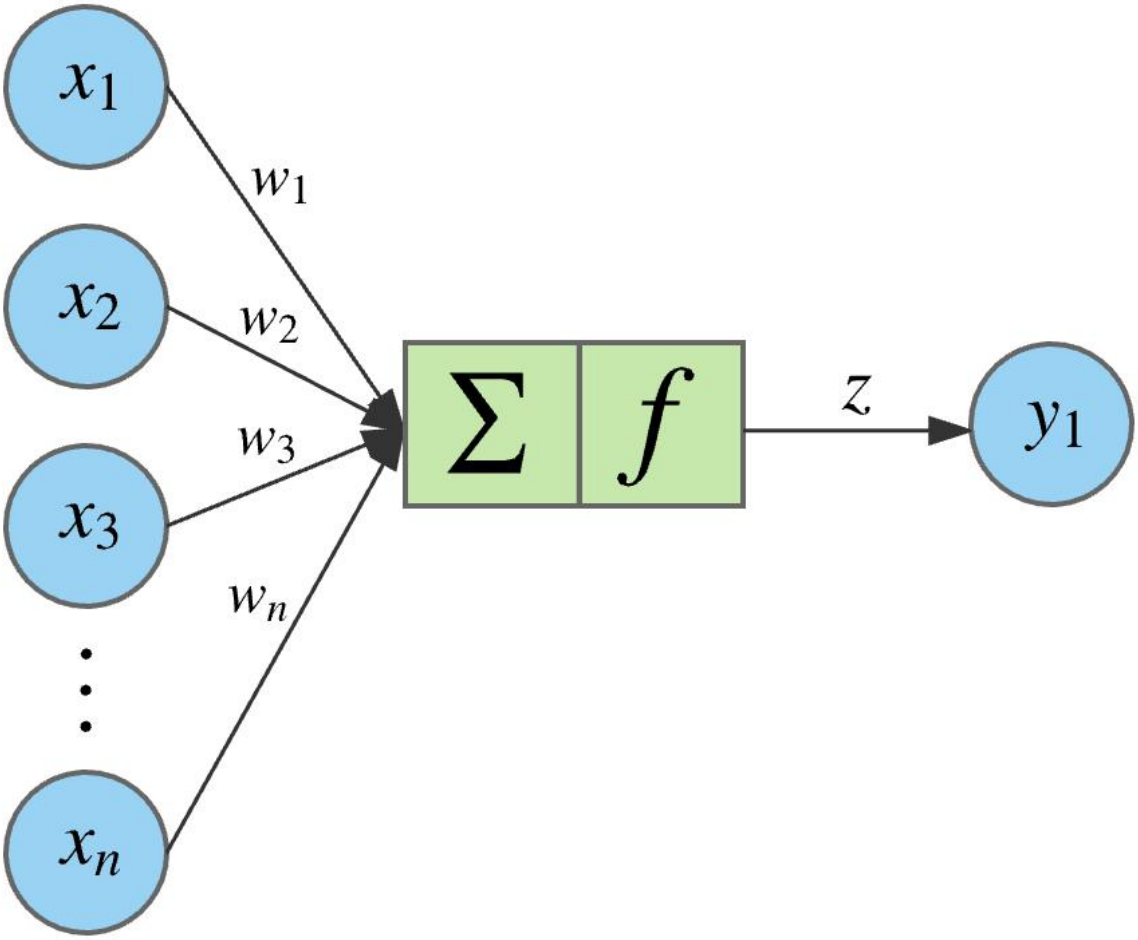


Figure 2

A given node takes the weighted sum of its inputs, and passes it through a non-linear activation function. This is the output of the node, which then becomes the input of another node in the next layer. The signal flows from left to right, and the final output is calculated by performing this procedure for all the nodes. Training this deep neural network means learning the weights associated with all the edges.
The equation for a given node looks as follows. The weighted sum of its inputs passed through a non-linear activation function. It can be represented as a vector dot product, where *n* is the number of inputs for the node

$$z = f(x \cdot w) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ z \in d_{1\times 1}$$

There are multiple types of neural network, each of which come with their own specific use cases and levels of complexity. The most basic type of neural net is something called a feedforward neural network, in which information travels in only one direction from input to output.

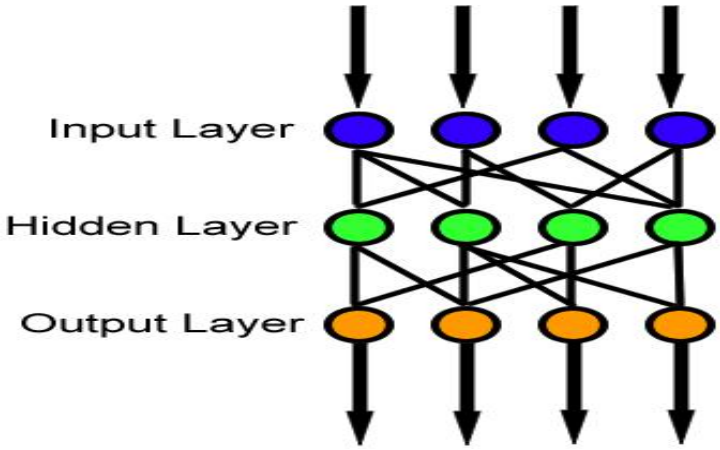


Figure 3(Feed Forward network)

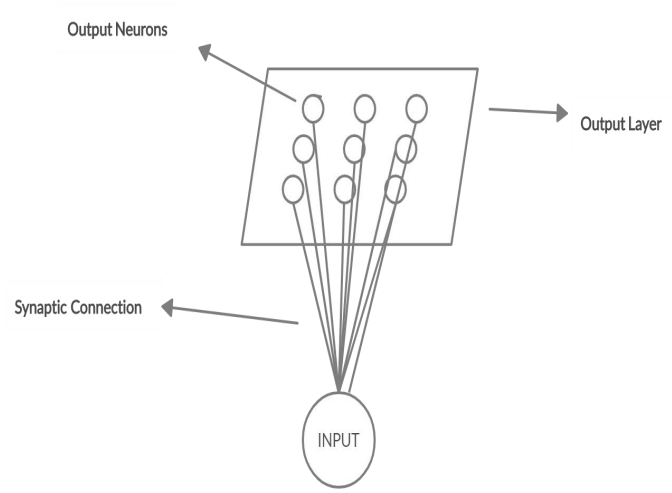
2. Self Organizing Maps:

A ***Self-organizing Map*** is a data visualization technique developed by Professor Teuvo Kohonen in the early 1980's. SOMs map multidimensional data onto lower dimensional subspaces where geometric relationships between points indicate their similarity.

The reduction in dimensionality that SOMs provide allows people to visualize and interpret what would otherwise be, for all intents and purposes, indecipherable data. SOMs generate subspaces with an unsupervised learning neural network trained with a competitive learning algorithm. Neuron weights are adjusted based on their proximity to "winning" neurons (i.e. neurons that most closely resemble a sample input).

Training over several iterations of input data sets results in similar neurons grouping together and vice versa.

2.1 Kohonen Self Organizing Maps:



Kohonen Self-Organizing feature map (SOM) refers to a neural network, which is trained using competitive learning. Basic competitive learning implies that the competition process takes place before the cycle of learning. The competition process suggests that some criteria select a winning processing element.

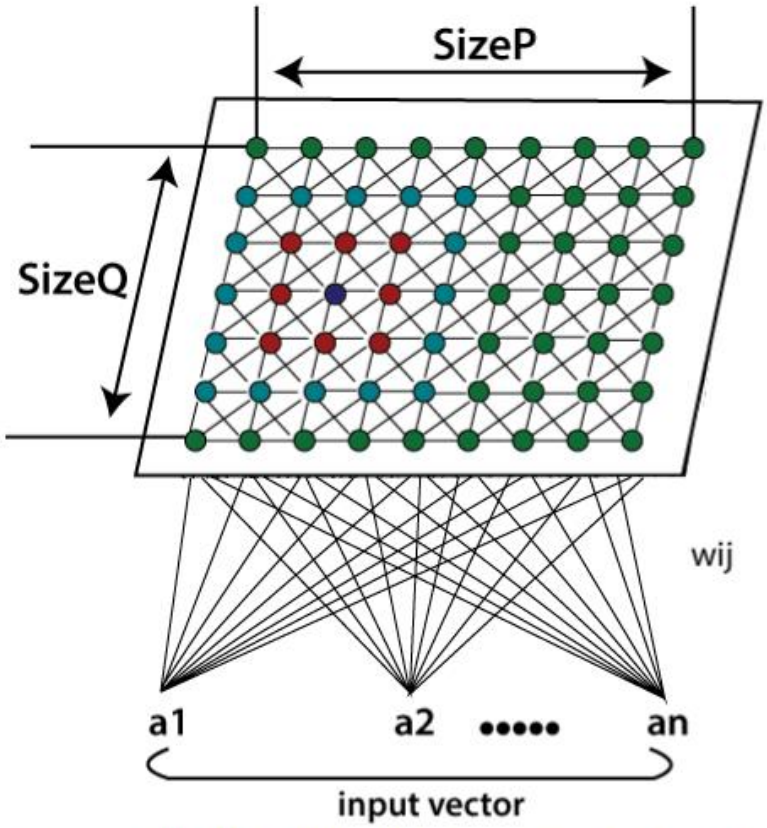
Kohonen Self Organizing Map is used in vector coding algorithms.It optimally places a fixed number of vectors(codewords) into a higher dimensional input space.

The essential processes in the formation of self organizing maps are :-

(1). ***Completion*** : For each input pattern the neurons in the output layer will determine the value of a function i.e. *Discriminant function.* Each neuron computes the discriminant function and the neuron with the largest discriminant function is the winner.

(2). ***Cooperation*** : The winning neuron will determine the topological neighbourhood of excited neurons.

(3). ***Synaptic Adaptation*** : It enables the excited neuron to increase their discriminant function in relation to the input vector.
The self-organizing map makes topologically ordered mappings between input data and processing elements of the map. Topological order implies that if two inputs are of similar characteristics, the most active processing elements answering to inputs that are located close to each other on the map. The weight vectors of the processing elements are organized in ascending to descending order. $W_i < W_{i+1}$ for all values of i or W_{i+1} for all values of i (this definition is valid for one-dimensional self-organizing map only).



A Kohonen model with the BMW in blue, the layer inside the neighborhood radius in light blue and red, and the nodes outside are green.

Figure 4

2.2 Algorithm:

1) Each node's weights are initialized.

2) A vector is chosen at random from the set of training data.

3) Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the **Best Matching Unit** (BMU).

4) Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.

5) The winning weight is rewarded with becoming more like the sample vector. The neighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.

6) Repeat step 2 for N iterations.

Best Matching Unit is a technique which calculates the distance from each weight to the sample vector, by running through all weight vectors. The weight with the shortest distance is the winner. There are numerous ways to determine the distance, however, the most commonly used method is the *Euclidean Distance.*
In steps it should be written like this :-
Step:1
Each node weight w_ij initializes to a random value.
Step:2
Choose a random input vector x_k.
Step:3
Repeat steps 4 and 5 for all nodes on the map.
Step:4
Calculate the Euclidean distance between weight vector w_{ij} and the input vector x(t) connected with the first node, where t, i, j =0.
Step:5
Track the node that generates the smallest distance t.
Step:6
Calculate the overall Best Matching Unit (BMU). It means the node with the smallest distance from all calculated ones.
Step:7
Discover topological neighborhood *βij(t) its radius σ(t)* of BMU in Kohonen Map.
Step:8
Repeat for all nodes in the BMU neighborhood: Update the weight vector w_ij of the first node in the neighborhood of the BMU by including a fraction of the difference between the input vector x(t) and the weight w(t) of the neuron.
Step:9
Repeat the complete iteration until reaching the selected iteration limit t=n.
Here, step 1 represents the initialization phase, while step 2 to 9 represents the training phase.
Where;
t = current iteration.
i = row coordinate of the nodes grid.
J = column coordinate of the nodes grid.
W= weight vector
w_ij = association weight between the nodes i,j in the grid.
X = input vector
X(t)= the input vector instance at iteration t
β_ij = the neighborhood function, decreasing and representing node i,j distance from the BMU.
σ(t) = The radius of the neighborhood function, which calculates how far neighbor nodes are examined in the 2D grid when updating vectors. It gradually decreases over time.

Appendix B (CRISP MODEL STEPS)

1) Business Understanding

A **pandemic** is an epidemic of an infectious disease that has spread across a large region, for instance multiple continents or worldwide, affecting a substantial number of people. A widespread endemic disease with a stable number of infected people is not a pandemic. Widespread endemic diseases with a stable number of infected people such as recurrences of seasonal influenza are generally excluded as they occur simultaneously in large regions of the globe rather than being spread worldwide.

The World Health Organization (WHO) previously applied a six-stage classification to describe the process by which a novel influenza virus moves from the first few infections in humans through to a pandemic. It starts when mostly animals are infected with a virus and a few cases where animals infect people, then moves to the stage where the virus begins to be transmitted directly between people and ends with the stage when infections in humans from the virus have spread worldwide.

In 2014, The United States Centers for Disease Control and Prevention introduced an analogous framework to the WHO's pandemic stages titled the Pandemic Intervals Framework. It includes two pre-pandemic intervals.

- Investigation
- Recognition

and four pandemic intervals,

- Initiation
- Acceleration
- Deceleration
- Preparation

It also includes a table defining the intervals and mapping them to the WHO pandemic stages.

COVID-19

A new strain of coronavirus which was first identified in Wuhan, Hubei province, China, in late December 2019, has caused a cluster of cases of an acute respiratory disease, which is referred to as coronavirus disease 2019 (COVID-19). According to media reports, more than 200 countries and territories have been affected by COVID-19, with major outbreaks occurring in central China, Iran, Western Europe and the United States. On 11 March 2020, the World Health Organization

characterized the spread of COVID-19 as a pandemic. As of 19 May 2020, the number of people infected with COVID-19 has reached over 4.89 million worldwide, of whom 1,908,111 have recovered. The death toll is 320,189. It is believed that these figures are understated as testing did not commence in the initial stages of the outbreak and many people infected by the virus have no or only mild symptoms and may not have been tested. Similarly, the number of recoveries may also be understated as tests are required before cases are officially recognised as recovered, and fatalities are sometimes attributed to other conditions. This was especially the case in large urban areas where a non-trivial number of patients died while in their private residences. It was later discovered that asymptomatic hypoxia due to COVID-19 pulmonary disease may be responsible for many such cases.

2) Data Understanding

The Covid-19 data set contains mostly continuous variables. This Data was last updated on April 15,2020.

The proper explanation of the data is given below.

```
In [114]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 212 entries, 0 to 211
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Country             212 non-null    object
 1   Total Cases         212 non-null    int64
 2   Cases               212 non-null    int64
 3   Total Deaths        212 non-null    int64
 4   Deaths              212 non-null    int64
 5   Total Recovers      211 non-null    float64
 6   Active              212 non-null    int64
 7   Total Cases/1M pop  212 non-null    float64
 8   Deaths/1M pop       212 non-null    float64
 9   Total Test          212 non-null    int64
 10  Tests/1M pop        212 non-null    int64
 11  Date                212 non-null    object
dtypes: float64(3), int64(7), object(2)
memory usage: 20.0+ KB
```

```
In [112]: df = pd.read_csv('datasets_602378_1084128_testingdataworldwide_April_15.csv')
```

```
In [113]: df.head()
```

Out[113]:

	Country	Total Cases	Cases	Total Deaths	Deaths	Total Recovers	Active	Total Cases/1M pop	Deaths/1M pop	Total Test	Tests/1M pop	Date
0	USA	619716	5830	27190	1143	47639.0	544887	1872.0	82.0	3151093	9520	15-Apr-20
1	Spain	177633	3573	18579	324	70853.0	88201	3799.0	397.0	600000	12833	15-Apr-20
2	Italy	165155	2667	21645	578	38092.0	105418	2732.0	358.0	1117404	18481	15-Apr-20
3	France	143303	0	15729	0	28805.0	98769	2195.0	241.0	333807	5114	15-Apr-20
4	Germany	133154	944	3592	97	72600.0	56962	1589.0	43.0	1317887	15730	15-Apr-20

3) Data Preparation

To start training the model we should normalize the values between 0 to 1. So, the model should work efficiently.

The full implementation of data preparation is given below.

```
In [72]: from sklearn.preprocessing import MinMaxScaler

In [76]: X_a = ['Total Cases', 'Cases', 'Total Deaths',
                 'Deaths', 'Total Recovers', 'Active','Total Cases/1M pop','Deaths/1M pop','Total Test','Tests/1M pop']
         X_arr = coronadata[X_a].values

In [77]: X_arr

Out[77]: array([[6.197160e+05, 5.830000e+03, 2.719000e+04, ..., 8.200000e+01,
                 3.151093e+06, 9.520000e+03],
                [1.776330e+05, 3.573000e+03, 1.857900e+04, ..., 3.970000e+02,
                 6.000000e+05, 1.283300e+04],
                [1.651550e+05, 2.667000e+03, 2.164500e+04, ..., 3.580000e+02,
                 1.117404e+06, 1.848100e+04],
                ...,
                [1.000000e+00, 0.000000e+00, 0.000000e+00, ..., 0.000000e+00,
                 0.000000e+00, 0.000000e+00],
                [1.000000e+00, 0.000000e+00, 0.000000e+00, ..., 0.000000e+00,
                 0.000000e+00, 0.000000e+00],
                [8.229500e+04, 4.600000e+01, 3.342000e+03, ..., 2.000000e+00,
                 0.000000e+00, 0.000000e+00]])

In [78]: scaler=MinMaxScaler(feature_range=(0,1))

In [79]: rescaledX=scaler.fit_transform(X_arr)

In [80]: np.set_printoptions(precision=3)

In [81]: rescaledX[0:5,:]

Out[81]: array([[1.   , 1.   , 1.   , 1.   , 0.612, 1.   , 0.171, 0.077, 1.   ,
                 0.082],
                [0.287, 0.613, 0.683, 0.283, 0.911, 0.162, 0.347, 0.374, 0.19 ,
                 0.11 ],
                [0.266, 0.457, 0.796, 0.506, 0.49 , 0.193, 0.249, 0.337, 0.355,
                 0.159],
                [0.231, 0.   , 0.578, 0.   , 0.37 , 0.181, 0.2  , 0.227, 0.106,
                 0.044],
                [0.215, 0.162, 0.132, 0.085, 0.933, 0.105, 0.145, 0.041, 0.418,
                 0.135]])

In [85]: np.isnan(rescaledX).any()

Out[85]: True

In [94]: rescaledX = np.nan_to_num(rescaledX)

In [95]: np.isnan(rescaledX).any()

Out[95]: False
```

We have rescaled the Data using preprocessing package of sklearn library.

To convert some non number values(eg. Null) we have used nan_to_num() function of numpy library.

4) Modeling

Now, For training the model we have used minisom library which is the implementation of Self Organizing Maps in Python.

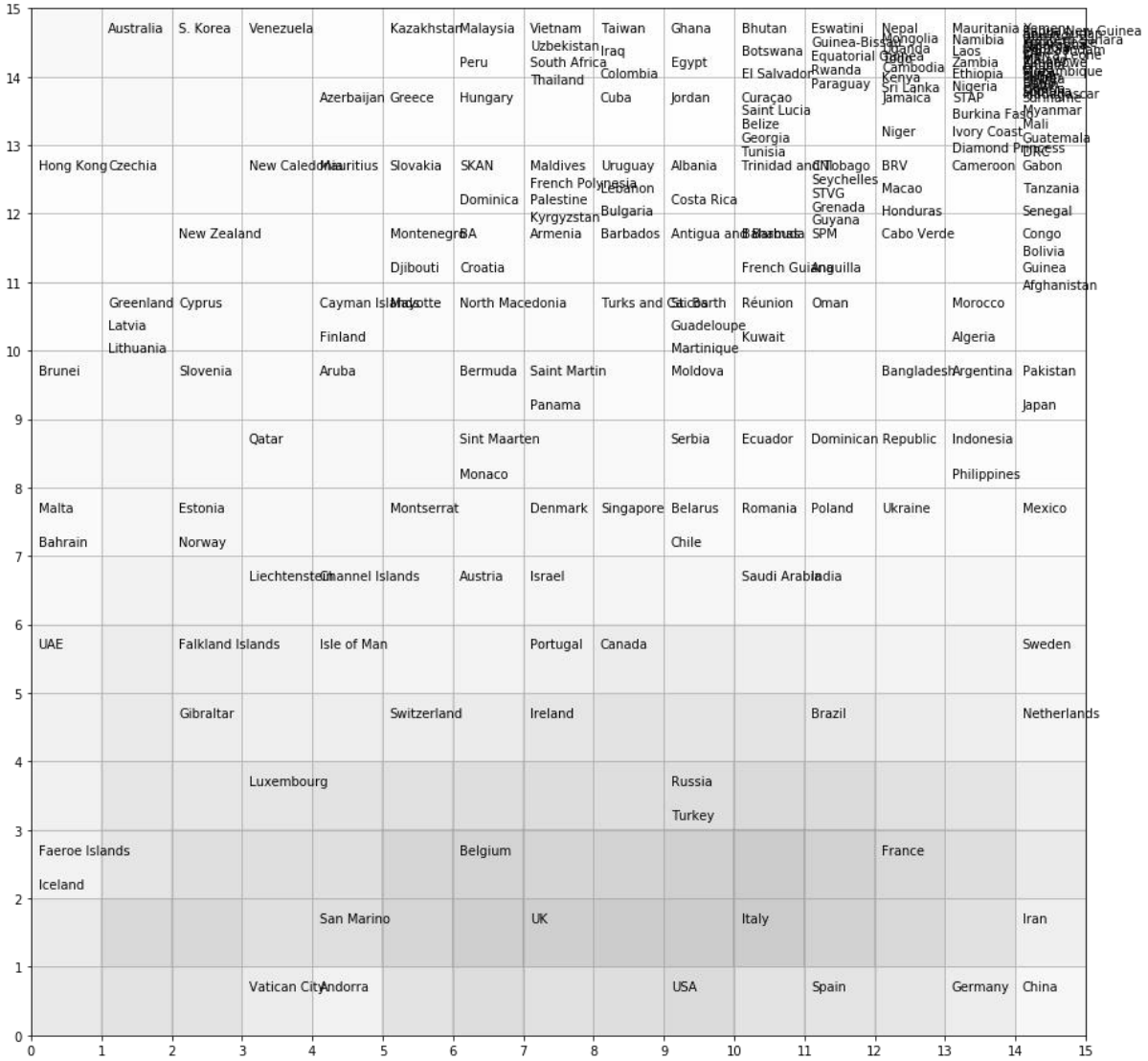
In [96]:
```
size = 15
som = MiniSom(size, size, len(rescaledX[0]),
              neighborhood_function='gaussian', sigma=1.5,
              random_seed=1)

som.pca_weights_init(rescaledX)
som.train_random(rescaledX, 1000, verbose=True)
```

```
 [  848 / 1000 ]  85% - 0:00:00 left

 [ 1000 / 1000 ] 100% - 0:00:00 left
 quantization error: 0.036437609790556616
```

After training the model we have plotted the countries in a map with matplotlib and minisom.

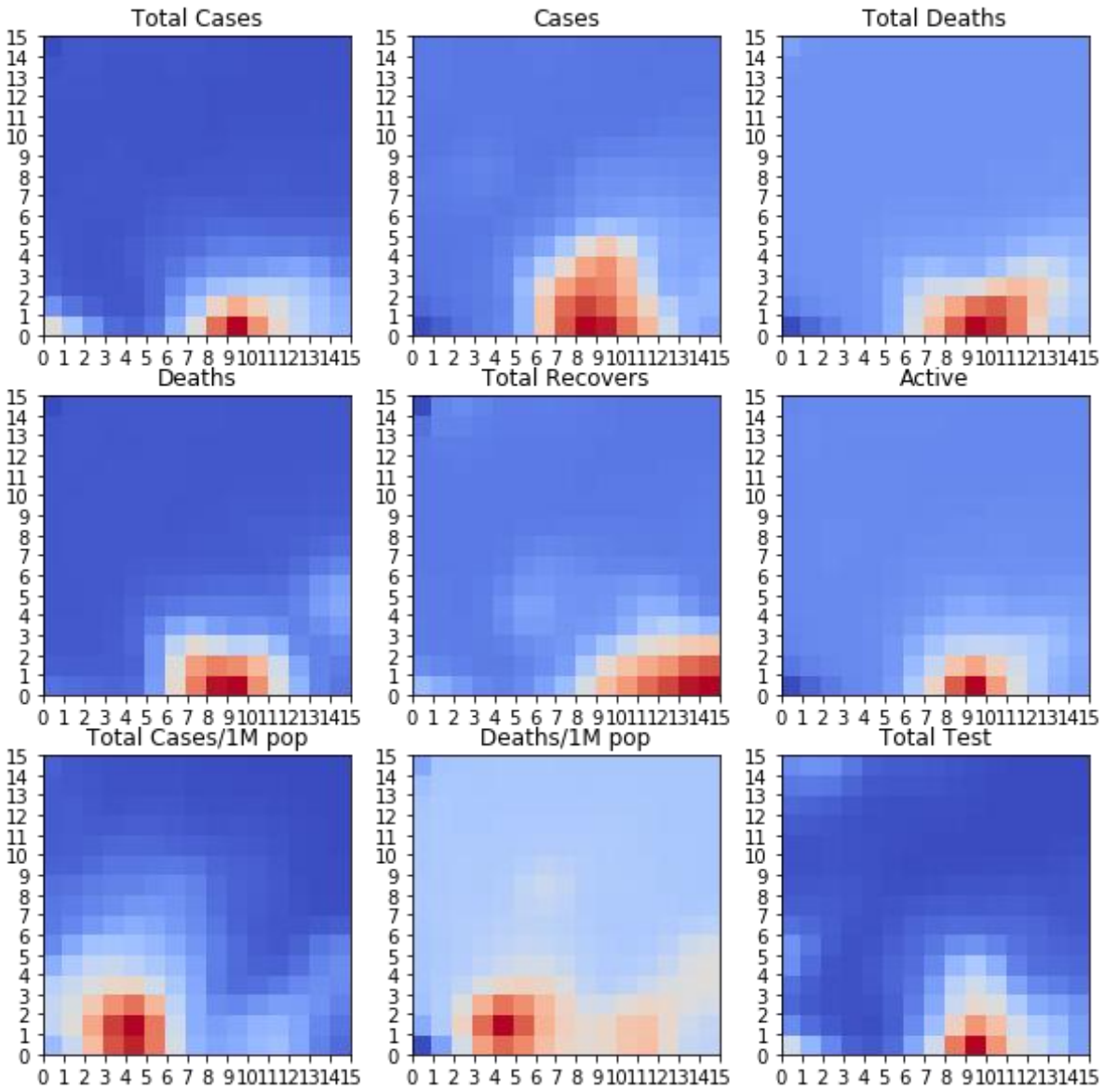
Now, the dark area in the map represents the countries which are most affected by the pandemic.

In [100]:
```
def shorten_country(c):
    if len(c) > 20:
        return country_codes[c]
    else:
        return c

country_map = som.labels_map(X, coronadata.Country)

plt.figure(figsize=(15, 15))
for p, countries in country_map.items():
    countries = list(countries)
    x = p[0] + .1
    y = p[1] - .3
    for i, c in enumerate(countries):
        off_set = (i+1)/len(countries) - 0.05
        plt.text(x, y+off_set, shorten_country(c),fontsize=10)
plt.pcolor(som.distance_map().T, cmap='gray_r', alpha=.2)
plt.xticks(np.arange(size+1))
plt.yticks(np.arange(size+1))
plt.grid()

legend_elements = [Patch(facecolor=clr,
                         edgecolor='w',
                         label=l) for l, clr in category_color.items()]
plt.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1, .95))
plt.show()
```



After this we have plotted individual planes for each features.

```
In [98]: W = som.get_weights()
         plt.figure(figsize=(10, 10))
         for i, f in enumerate(feature_names):
             plt.subplot(3, 3, i+1)
             plt.title(f)
             plt.pcolor(W[:,:,i].T, cmap='coolwarm')
             plt.xticks(np.arange(size+1))
             plt.yticks(np.arange(size+1))
         plt.tight_layout()
         plt.show()
```



At last we have plotted most relevant plane for all the features to identify which features have most affect in the model.

```
In [72]: from sklearn.preprocessing import MinMaxScaler
```

```
In [76]: X_a = ['Total Cases', 'Cases', 'Total Deaths',
                 'Deaths', 'Total Recovers', 'Active','Total Cases/1M pop','Deaths/1M pop','Total Test','Tests/1M pop']
         X_arr = coronadata[X_a].values
```

```
In [77]: X_arr
```

```
Out[77]: array([[6.197160e+05, 5.830000e+03, 2.719000e+04, ..., 8.200000e+01,
                 3.151093e+06, 9.520000e+03],
                [1.776330e+05, 3.573000e+03, 1.857900e+04, ..., 3.970000e+02,
                 6.000000e+05, 1.283300e+04],
                [1.651550e+05, 2.667000e+03, 2.164500e+04, ..., 3.580000e+02,
                 1.117404e+06, 1.848100e+04],
                ...,
                [1.000000e+00, 0.000000e+00, 0.000000e+00, ..., 0.000000e+00,
                 0.000000e+00, 0.000000e+00],
                [1.000000e+00, 0.000000e+00, 0.000000e+00, ..., 0.000000e+00,
                 0.000000e+00, 0.000000e+00],
                [8.229500e+04, 4.600000e+01, 3.342000e+03, ..., 2.000000e+00,
                 0.000000e+00, 0.000000e+00]])
```

```
In [78]: scaler=MinMaxScaler(feature_range=(0,1))
```

```
In [79]: rescaledX=scaler.fit_transform(X_arr)
```

5) Evaluation

We have trained the model two times first time on the raw data and second time on the normalized data. It has been observed that quantization error has significantly decreased in second time.

This is more explained by the code below.

```
In [110]: size = 15
          som = MiniSom(size, size, len(X[0]),
                        neighborhood_function='gaussian', sigma=1.5,
                        random_seed=1)

          som.pca_weights_init(X)
          som.train_random(X, 1000, verbose=True)

           [  741 / 1000 ]  74% - 0:00:00 left

           [ 1000 / 1000 ] 100% - 0:00:00 left
           quantization error: 0.3622648600389903
```

FIRST TIME

```
In [96]: size = 15
         som = MiniSom(size, size, len(rescaledX[0]),
                       neighborhood_function='gaussian', sigma=1.5,
                       random_seed=1)

         som.pca_weights_init(rescaledX)
         som.train_random(rescaledX, 1000, verbose=True)

          [  848 / 1000 ]  85% - 0:00:00 left

          [ 1000 / 1000 ] 100% - 0:00:00 left
          quantization error: 0.036437609790556616
```

SECOND TIME

It is clearly indicated above that Quantization error in first time is far higher then that of second time.