Developing Self-Organising Maps for categorical data

Steve Hong, Ferran Espuny Pujol

August 11, 2022

Abstract

Self-organising map, first developed by Kohonen, is a clustering method that helps to display very large, high-dimensional datasets in a low-dimensional space. It is a special variation of unsupervised neural networks, where processing units, or neurons, are arranged in a 2-dimensional grid and after certain iterations of update, observed data could be projected onto the grid so that clusters of data can be revealed while preserving the topology of the data set. However, the original SOM algorithm is only designed for numerical data, which makes it extremely limited for most datasets nowadays that contains both numerical and categorical data. This research will therefore investigate the available extensions of the SOM for dealing with categorical data and compare their effectiveness, then decide the most feasible way to update the weight of the neurons.

1 Introduction

1.1 Explanation of the original SOM

SOM [1] is used for visualising high-dimensional datasets by transforming them into groups of observations that have similar characteristics, which can be displayed on a 2D map. SOM is made of a grid of neurons as follow:

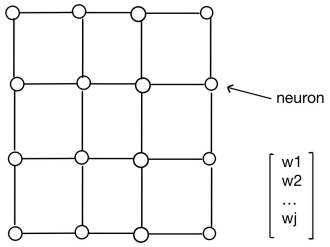

Figure 1: A neuron grid.

Each of these neuron carries a set of weight, which are initialised as vectors of random values. It is crucial that the final output would be the grid being bent and twisted so that it reflects the clusters of data, and this is done by changing these weight values in iterations, so that when the whole process stops, these weights will each stand for the average values of the parameters of the cluster of data.

Each of the iteration works as follow. Each of the iteration works as follow. Firstly, an input data can be randomly selected and introduced to the grid of neurons. Choosing the centre of a cluster happens on a competitive basis. The neuron that is closest to that input, determined by the shortest Euclidean distance between the weight and the input vector, will be chosen as the winning neuron (or Best Matching Unit). This is shown in Figure 2.

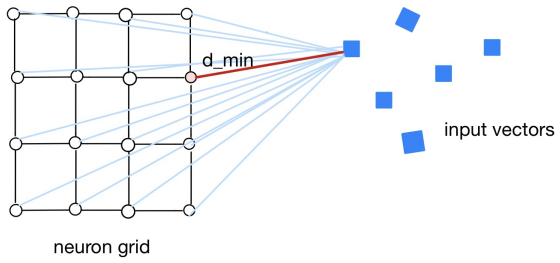

Figure 2: How a winning neuron is determined.

The winning neuron, as well as its neighbours, will then be moved towards that input data. The further the neighbour neurons are, the less they are moved. These are done by updating the weight vector of the winning neuron and the neighbours, shown in Figure 3.

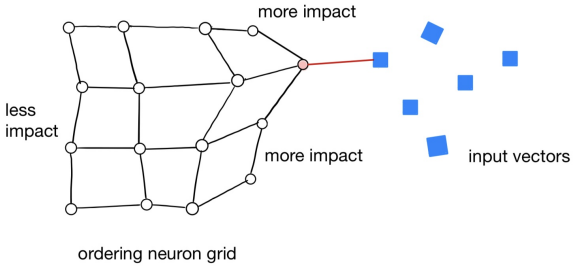

Figure 3: How all the neurons are moved after each update.

This process is repeated for a pre-determined number of iterations, or until there is no major changes in the shape of the grid. The outcome is an ordered grid, where neurons that are close to each other on the grid have similar features. This can be shown in Figure 4:

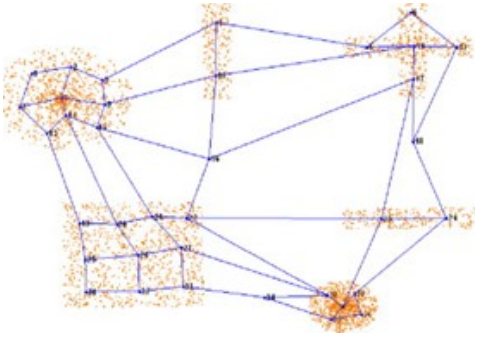

Figure 4: The ordered grid after a number of iterations.

1.2 Limitations of the original online SOM

Incompatible with categorical data: It would not be possible for the original SOM algorithm to update the neuron weights for categorical data. Let's consider a simple example of a neuron whose weight vector includes a variable that indicates whether a person is ethnically White or Black. SOMs also uses minimum geometric distance to determine the winning neuron, where the use of numerical inputs are appropriate. It is, however, not possible to define such distances for non-ordinal categorical data because codifying them would inaccurately represent the meaning of the data. For example, race data such as (White, Asian, Black, American Indian, Other Pacific Islander) could not be replaced by (1,2,3,4,5), because the pairwise distance between them are not the same.

Online algorithm: Why use batch instead of online? (To be updated)

1.3 Available Extensions

There have been several attempts to rectify such short comings of the original SOM.

1.3.1 Binary codes

A typical approach to process categorical data is to convert them into binary variables. For variables with only two values, namely male/female, a variable Male that takes 1 for male and 0 for female could be introduced [2].

For non-ordinal variables with more than two values, such as the race example above, multiple binary variables could be formed: White {0,1}, Asian {0,1}, Black {0,1}, American Indian {0,1}, Other Pacific Islander {0,1}.

It will be a good idea to set a numerical constraint onto dummy variables. In this case White + Asian + American Indian + Other Pacific Islander = k, where k is a constant, meaning that each person can only carry a fixed number of ethnicities.

1.3.2 NCSOM (Batch SOM algorithm)

Compared to the original online algorithm, this extension is implemented in batch, which means all data are considered at once every time the model is trained [3].

Because of the unknown geometric distance between the categorical values of the input vector and the weight vector, a direct matching procedure could be used to quantify the distance: 0 if they are in the same category and 1 otherwise. To ensure that all features have equal influence on the total distance, numerical data are standardised before calculation.

1.3.3 GSOM (Generalised Self-organising Map)

The GSOM approach enables direct calculation of distance between categorical variables by introducing distance hierarchy. The structure is made of nodes and link weights, higher-level nodes account for more general concepts, while lower-level nodes account for more specific concepts [4]. The distance between two concepts at leaf nodes is defined as the total link weight between those two leaf nodes. Link weights are assigned by domain specialists.

1.3.4 FMSOM (Frequency neuron Mixed Self-Organising Map)

The FMSOM approach is built upon the NCSOM model. While keeping the original approach to numerical data, the FMSOM introduces a frequency table for each value of categorical data to extend neuron prototypes. The distance between the numerical features is calculated using geometric distance such as Euclidean that between categorical features is calculated using probability.

2 Methodology with NCSOM

2.1 Explaining notations

- x is the input vector: $x_i = [x_{i1}, x_{i2}, \cdots, x_{id}]$ where $i = 1, 2, \cdots, N$ and d is the number of feature and N is the number of observations.

- w is the weight vector: $w_j = [w_{j1}, w_{j2}, \cdots, w_{jd}]$ where $j = 1, 2, \cdots, m \times n$ and d is the number of features. m and n are the dimensions of the grid.
- c_j is the vector of indices of Best Matching Units (BMU), which is the neuron that has the smallest distance from the current input vector.
- Within d features, there are p numerical features and k categorical features. The possible outcomes of each categorical feature is $\{\alpha_k^1, \alpha_k^2, \cdots, \alpha_k^{n_k}\}$, where n is the possible number of outcomes at k^{th} feature
- h_{c_j} is the vector of neighbourhood function values between the BMU and the rest of the neurons. h_{c_j} is mathematically defined below, where where r is the coordinates of the neurons on the $m \times n$ grid, and t is the t^{th} iteration.

$$h_{c_j} = \exp(-\frac{(\|r_j - r_{c_j}\|)^2}{2 \times \delta^2(t)})$$

2.2 Dissimilarity Measure

The distance D between the input vector x_i and w_j is defined to be the combination of the Euclidean distance squared for the numerical variables and either 0 or 1 for the categorical variables [3].

$$D(x_i, w_j) = \sum_{l=1}^{p}(x_{il}, w_{jl})^2 + \sum_{l=p+1}^{d} \delta(x_{il}, w_{jl})$$

where:

$$\delta(x_{il}, w_{jl}) = \begin{cases} 0, & \text{if } x_{il} = w_{jl} \\ 1, & x_{il} \neq w_{jl} \end{cases}$$

2.3 Update weight vectors

The batch update process can be divided into 4 steps [5] :

1) Input in parallel the input vector x_i to all the neurons

2) The distance between each input vector and all neurons can be calculated using the dissimilarity formula introduced in 2.2. The list of indices of BMUs could be found by appending step-by-step the index of the shortest neuron to the data.

3) There are two parts of the weights: Numerical and Categorical features, which could be updated using NCSOM [3].

- **Numeric Features**:
$$w_{pk}(t+1) = \frac{\sum_{i=1}^{n} h_{c_i p} \times x_{ik}}{\sum_{i=1}^{n} h_{c_i p}}$$
- **Categorical Features**: The update for categorical operates in an order of: category - relative frequency - category. The new value for the categorical feature is one with the highest relative frequency, calculated with the formula below, compared to the sum of the rest of the possible values in that variable. For example, if there are three possibilities for the Ethinicity feature $Black, White, Asian$, with the relative frequencies of {0.6,0.1,0.3 . Black would be updated to the Ethnicity variable of that neuron, because it has the F(Black, w_1) value that is larger than the sum of the other 2.

$$F(\alpha_k^r, w_{pk}) = \frac{\sum_{i=1}^{n}(h_{c_i p}|x_{ik} = \alpha_k^r)}{\sum_{i=1}^{n} h_{c_i p}}$$

4) Repeat the three steps for a predefined number of iterations, or until there is negligible change in the neighbourhood function value, which will also be defined.

3 Pseudocode

```
Step 1: Initialise the weights  and normalise input data

#Random generation of the weights:
#We need to randomise the numerical part and the categorical part, then merge them

define j,p,k
define m = [ ]

#Numerical
for each neuron in range j:
m.append(np.random.rand(p,1))

#Categorical
#extract the possible categories for each variable (using unique() function on each variable)
#a library can be found to randomise the outcome of each variable. If there is none available,
#one way could be to make a rand_cat() function that randomises integer (0, number of categories) the
#the returned index to the list of unique possibilities from above.

for mj in m:
cat = rand(k) for k in range of (d-p)
append cat to each neuron

#Step 2: Calculate distances and BMU for each observation

#Define a function to calculate the distance between x and m by using index
Def distance(data, neuron, no_of_numvar, no_of_var ):
num_dist = (data[0, no_of_numvar] { neuron[0, no_of_numvar])2
cat_dist = 0
for counter in range (no_of_numvar + 1, no_of_var):
if data[no_of_numvar+1] == neuron [no_of_var]:
cat_dist += 0
else:
cat_dist += 1
return num_dist + cat_dist

#for each xi, compute d(xi,mj) for all values of m. Create tracker variables to track the minimum val
c_list = [ ]
min_dist = 0
for xi in x:
dist = distance(xi, mj) for mj  in m
if dist < min_dist:
c_list . append (index(mj))
min_dist = min(dist, min_dist)


#Step 3: Update the weights using all data (Batch Algorithm)

#Compute a list of neighourhood function values for each value of m using the list of c

#Numerical
for each neuron on the grid:

find its distances (h)  on the grid from the winning neurons from the list, then divide by
a function of time/number of iteration. The distance is calculated in Cartesian form. Store
```

```
these values in a list.

denominator =  the sum of these values of h
numerator_num = the sum of these values multiplied with the list of input vectors
new_weight_num  = numerator/denominator


#Categorical

def f_value(name_of_variable, name_of_outcome):

filter out the winning neurons (referenced from above) whose
name_of_variable variable = name_of_outcome. Make them a list

find the distances on the grid between these winning neurons and the neuron
to be updated. Store these distance values in list_1

find the distances on the grid between all winning neurons and the neuron
to be updated. Store these distance values in list_2

numerator_cat = sum of  the values of h from list_1
denominator = sum of all the values of h from list_2

return numerator_cat/denominator

for each neuron on the grid:
for each categorical variable (indices from p+1 to d):
f_list = [ ]
f = f_value(name_of_variable, name_of_outcome) and flist.append(f) for each of its
possible outcome

for each of the f values from f_list:
if it is bigger than sum of the rest:
update the weight to the name_of_outcome
else:
keep the current weight
```

4 Result and future work

Due to the time constraint of this project, the implementation process only progressed until integrating the batch algorithm to the SOM package, which could be found here. The trained network returned reasonably similar results to that of the standard online algorithm of the SOM. The three clusters of seed labels are clearly displayed, yet they are shown in a symmetrically mirrored structure, which does not pose any major problems to the model. They could be compared as below:

(a) Online SOM

(b) Batch SOM

Figure 5: Trained Online SOM and Batch SOM

Further progressions of this project will include:

- Modify the Distance and Update function so that they are compatible with categorical data. This can be implemented using the Pseudocode written in Section 3, using the NCSOM framework.
- Apply the completed algorithm on the Seed dataset, then on the Clinical Operation dataset and produce insights into the clusters displayed by the neuron grid.

References

[1] Kohonen, T. (2001) Self-Organizing Maps. 1st ed. Finland: Springer Berlin, Heidelberg.

[2] Carmelo del Coso, Diego Fustes, Carlos Dafonte, Francisco J. Nóvoa, José M. Rodríguez-Pedreira, Bernardino Arcay (2015) Mixing numerical and categorical data in a Self-Organizing Map by means of frequency neurons, Applied Soft Computing, Volume 36, Pages 246-254, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2015.06.058.

[3] Chen, N., Marques, N.M. (2005). An Extension of Self-organizing Maps to Categorical Data. EPIA.

[4] Lebbah, Mustapha Benabdeslem, Khalid. (2010). Visualization and clustering of categorical data with probabilistic self-organizing map. Neural Computing and Applications. 19. 393-404. 10.1007/s00521-009-0299-2.

[5] Matsushita, H. and Nishio, Y., (2010) Batch-Learning Self-Organizing Map with Weighted Connections Avoiding False-Neighbor Effects. WCCI 2010 IEEE World Congress on Computational Intelligence,.