Using Self-Organizing Maps on the Occupancy Dataset
Project 2 Step 1

Allina Dolor, James Robinson, Cherian Thomas, and Matthew Wise

Abstract

The Occupancy Detection dataset can potentially be used in order to predict whether or not a room is occupied by an individual by using its data about temperature and light levels. To determine if predicting occupancy is feasible with the dataset's provided attributes and items, self-organizing maps were used with the data to see if it clusters to match the occupancy. Using Orange2 with Python 2.7 as our primary tools, the data was previously preprocessed to remove outlier and normalize the continuous attributes. The self-organizing maps algorithms provided by Orange2 were then used to generate and reveal information about the data. Ultimately, we found that self organizing maps were an effective solution for the representation of the Occupancy Detection dataset due to the ease of predicting occupancy from the representation.

1. Background

The preprocessed occupancy detection dataset contains five continuous attributes of interest: Temperature, CO2, Humidity, Light, and HumidityRatio. Five attributes means that the self-organizing map's architecture would have a dimensionality of five. While the preprocessed dataset also contains a Date attribute, it was ignored due to it not being considered an interesting or potentially revealing part of the dataset. The values occurring in the date attribute were incremental and appeared to be supplied by the original creator as a meta attribute to keep track of the items. Due to this judgment, the dates were removed to avoid having them affect the results of the self-organizing map.

For this self-organizing map, parameters were set to both provide strong clustering and mitigate the processing time needed to generate euclidian distances and process the iterations of the self-organizing maps. After performing several experiments with generating self-organizing maps and viewing the results, the following parameters were chosen to induce the most accurate clusters for representing the original dataset's target attribute.

1.1. Topology

The self-organizing map uses a 2D hexagonal topology. While changing between a rectangular and hexagonal topology would not affect the results to a measurable degree, using a two-dimensional topology was preferred over a one-dimensional topology as it would be less prone to converging over iterations.

1.2. Number of neurons

The preprocessed occupancy dataset contains 5 different attributes for 18854 different items, so the number of neurons being used needs to take this into consideration. After a few experiments, we settled on using 400 neurons organized into 20 rows and 20 columns. 400 neurons, while an arguably low amount for the amount of items in the dataset, should be able to adequately spread itself across the 18854 data points with a smaller risk of any single data point becoming attached to single neurons. More could have been used to reduce potential error, but the increased algorithm processing times required to account for those extra neurons would have limited the time needed to experiment and analyze the clustering afterward.

1.3. Neighborhoods

The self-organizing map was initialized to have neighborhoods of radius 3 with a gaussian function. A gaussian function, unlike the bubble function, has it so neurons further away from winning neurons are less affected than those closer to winning neurons. The gaussian function was expected to give the neurons a better ability to spread out evenly, potentially forming more accurate clusters as the self-organizing map went through its iterations. Due to this, the gaussian function was selected for the majority of experiments. With each iteration of the self-organizing map, these neighborhoods decrease in size linearly to a value of 1.

1.4. Learning rate

The default learning rate in Orange2 is set to 0.05 with no straight-forward way of modifying the value. In order to set the learning rate for neurons with custom values that could be more effective with the occupancy dataset, a small script written in python was used. This script directly changed the value in the source code, which gave a workable method of choosing learning rates for the different self-organizing map experiments. Learning rate values ranged between 0.01 and 0.10 for the experiments.

For the learning rate reduction function, Orange uses the following function to calculate learning rates, starting from the initially set value and decreasing to 0 in the final epoch:

```
def alpha(self, iter):
    iterations = len(self.data)*self.epochs
    return (1-float(iter)/(iterations-1))*self.learning_rate
```

Orange2 does not provide a clear alternative to this function; therefore, this function was used for all experiments on the dataset.

1.5. Number of Epochs

Throughout the process of experimenting with self-organizing maps on the occupancy dataset, the number of epochs performed for each experiment stayed mainly within 300 and 1000 epochs. Using values around 1000 started to provide diminishing changes to the final output for the self-organizing map algorithm while also increasing processing times significantly. Because of this, the number of epochs used stayed mostly at 300.

2. Experiments

Determining the number of clusters expected depends mostly on the specifics of the dataset involved. For our dataset, the occupancy target attribute demands 2 clusters from the self-organizing map because it is a binary attribute, containing either a value of 1 or 0 representing occupied or not occupied respectively. We expect the clusters formed by the self-organizing map to reflect these values coming from the dataset.

To analyze the performance of the SOM, we will run different experiments that pertain to changing the value for the learning rate, neighborhood size, and number of neurons. All parameters have a direct influence on the results of the self-organizing map.

2.1 Learning Rate

For the experiment, the learning rate will be adjusted a value lower (0.01) and a value higher (0.10) than the default set value of 0.05. The algorithm uses a reduction function that gradually decreases the value over time, bringing it close to zero during the final iterations. Changes to the learning rate value affect how significantly neurons in the self-organizing map move with each iteration.

By manipulating the learning rate and viewing the results, we found a variation of values in the confusion matrix after the three separate values. Compared to 0.01 learning rate, the default 0.5 saw an increase in true negatives (0, 0) by a value of 15 and inversely there was a decrease in false positives by a value of 15 (1, 0). Comparing the two values again, the false negatives (0, 1) increased while the true positives decreased (1, 1).

We continued to set a higher learning rate to 0.10. The results yielded a decrease by 11 in true negatives (0, 0) and inversely yielded an increase by 11 in false positives (1, 0). Furthermore, there was also an decrease in false negatives by 9 however had an increase of 9 for the true positives (1, 1).

Ultimately, we can see a pattern between each of the three iterations; when there is a change in the value of true negatives, there is an inverse of false positives. Similarly, when there is a change of value between false negatives, there is an inverse change of values for true positives.

2.2 Neighborhood

Continuing with the experiments, the neighborhoods for neurons were tested with both a gaussian function and a bubble function. In contrast to the gaussian function, bubble function maintain a constant pull over neurons in the neighborhoods of the winning neuron, disregarding the degree of separation. Both functions use the set radius to determine which surrounding neurons are viable candidates for adaptation.

Between bubble and gaussian neighborhoods, both had identical performance in the performed experiments. Both the bubble and gaussian functions resulted in the self-organizing map creating two clusters, which is what was expected. This finding supports the idea that the neighborhood function used does not have a measurable impact on this particular dataset.

2.3 Number of Neurons

The number of neurons used in the self-organizing map experiments did have a fair effect on the results, but only to a certain extent. Switching between 100 and 400 neurons made the self-organizing map more accurate. When using 400 neurons, the self-organizing map correctly formed the expected two clusters whereas using 100 neurons incorrectly formed eight.

Going beyond 400 neurons to 900 neurons, the self-organizing map was also able to correctly form two clusters. However, there was no evidence of using 900 neurons being better performing than 400 in creating these clusters. Contrarily, the 400 neuron experiment was able to form the same clusters, but with a much smaller impact on time consumed to run the algorithm. Because of this occurrence, any amount of neurons close to 400 neurons should be good for this dataset.

3. Results

The Orange self-organizing map visualization tool was used to view all of the experiments. However, in order to gain better detail into specific experiments, the output neurons were further processed by both a convolutional matrix and a hierarchical k-Means algorithm with silhouette scoring to find the optimal clusters. The k-Means algorithm was able to more accurately show how the neurons formed clusters after being exposed to the dataset over solely using a convolutional matrix.

The results for the SOM visualization and confusion table are provided in a separate folder container the different experiments, as well as an analysis of the effects of accuracy of each variation.

4. Discussion

Self-organizing maps can be used to analyze relationships between input patterns by creating topologies. In addition, it may be used to visualize the relationships between input data. We found that SOMs are overall quite well suited to the task of occupancy prediction. The default hyperparameters produced a SOM capable correctly predicting occupancy in over 99% of cases. Slight improvements in accuracy were observed with severe diminishing returns with respect to time complexity with increased neuron count and number of iterations, and similarly, slight reductions in accuracy were observed with decreases in those quantities. Some other hyperparameters, such as neighborhood size and reduction function, could shift the misclassifications towards one category or another but had little overall effect on accuracy. Lowering the learning rate had little meaningful effect on accuracy but raising it has a significant negative effect, likely because the neurons couldn't be moved with sufficient precision to reach their proper places.