Dan Usman

COSC 4P80 Assignment 2 – Classifying Electric Motors Using Self-Organizing Maps

Introduction

Electric motors are an essential part in industry. Motors can be installed and run for years, but eventually they do fail. Part of the issue is predicting when a motor is about to fail or has failed. Given a sample of 53 motors where their fluctuations were recorded as a time series, I'm going to make use of Self-Organizing Maps to differentiate between good and bad motors. For this experiment, I used Holdout Validation to split our sample data of 53 into two sets; training and testing. 85% of the entire sample data was used for training while the other 15% was used for testing.

Self-Organizing Maps

Self-Organizing Maps is an array of N x N weight vectors. Each vector consists of the same amount of weights as there are attributes in an input data vector. In a simple sentence, Self-Organizing Maps make use of vector quantization to move its weight vectors closer to the exposed input data vector. The unique thing about SOMs is that it also allows the updated vector to influence the neighboring vectors. For this implementation a 15x15 lattice was used.

Initialization

Each weight inside a vector was randomly initialized between the max and min ranges of the observed attributes in the input data vectors within our training set.

Training

A random sample is chosen from the training set. We find the Euclidean distance between the chosen sample and all the vectors in the 15x15 lattice. The vector with the least distance is chosen to be the Best Matching Vector. Each weight is updated by a certain percentage (learning rate) of the difference between the respective weight (weight vector) and attribute (input data vector).

After the Best Matching Vector has been determined all the other weight vectors in the lattice needs to be updated. All the other weight vectors are updated the same way as the Best Matching Vector, however, we must also apply some influence. To determine our influence, we experimented with the Gaussian and the Mexican Hat function. The function makes use of Euclidean distance in the lattice between each weight vector and the Best Matching Vector and the radius around the Best Matching Vector. As training progresses, the learning rate and the radius decreases by some factor.

Results

All results are averaged from 100 runs

Gaussian Radial Function

Parameters					Accuracy			
#	Learning Rate	Max Epochs	LearningRate Decay Constant	Radius Decay Constant	L30fft16	L30fft25	L30fft32	L30fft64
1	0.1	1000	2	1	.77	0.76	0.74	0.77
2	0.3	1000	2	1	0.77	0.75	0.74	0.76
3	0.5	1000	2	1	0.77	0.76	0.75	0.71
4	0.1	1000	2	1.5	0.77	0.77	0.8	0.75
5	0.1	1000	2	2	0.78	0.77	0.79	0.77
6	0.1	1000	2	4	0.78	0.77	0.78	0.73
7	0.1	1000	1.5	2	0.81	0.79	0.81	0.75
8	0.1	1000	0	2	.79	0.77	0.80	0.75
9	0.6	1000	3	2	0.79	0.73	0.78	0.76

Mexican-Hat Radial Function

Parameters					Accuracy			
#	LearningRate	Max Epochs	LearningRate Decay Constant	Radius Decay Constant	L30fft16	L30fft25	L30fft32	L30fft64
10	0.1	1000	2	1	.75	0.72	0.73	0.68
11	0.1	1000	2	3	0.74	0.76	0.75	0.75
12	0.1	1000	2	4	0.72	0.79	0.77	0.76
13	0.1	1000	1	4	0.76	0.76	0.81	0.75
14	0.3	1000	1	4	0.63	0.63	0.61	0.64

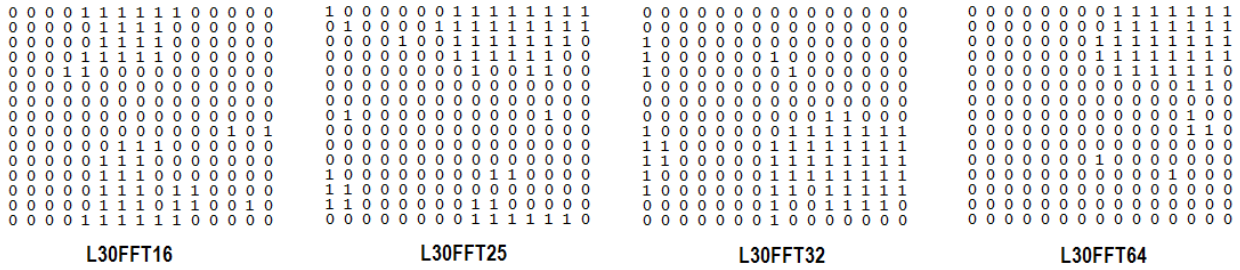

Figure 1 – seed: 2222, sample: 7

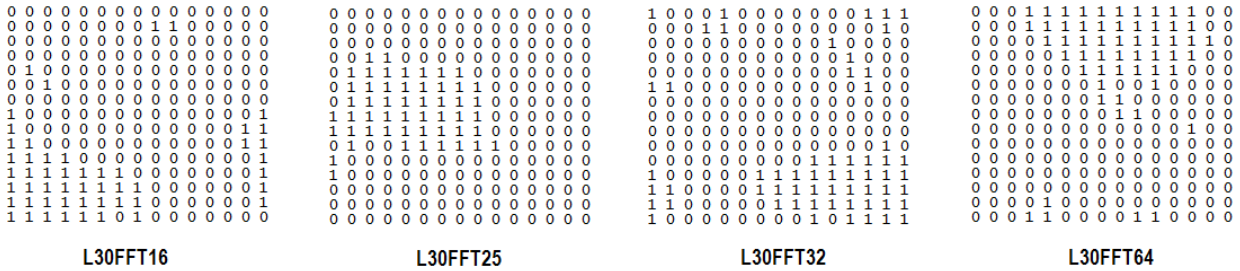

Figure 2 – seed: 2222, sample: 3

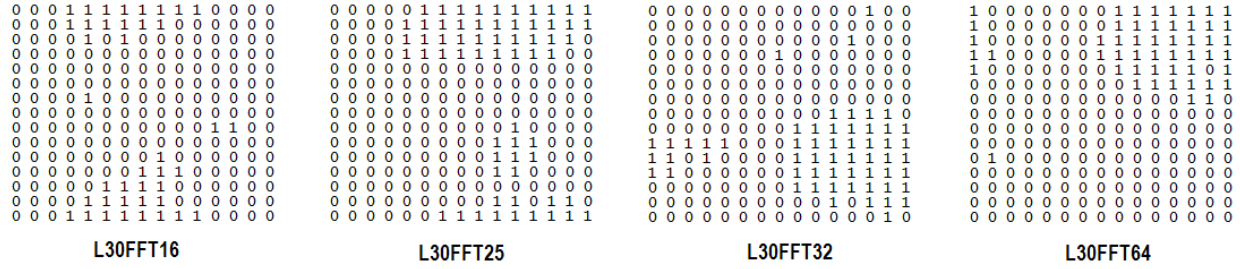

Figure 3 – seed: 2222, sample: 1

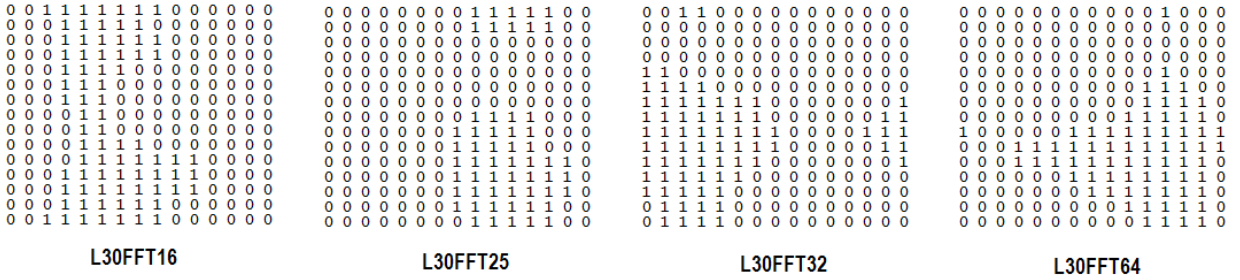

Figure 4 – seed: 2222, sample: 10

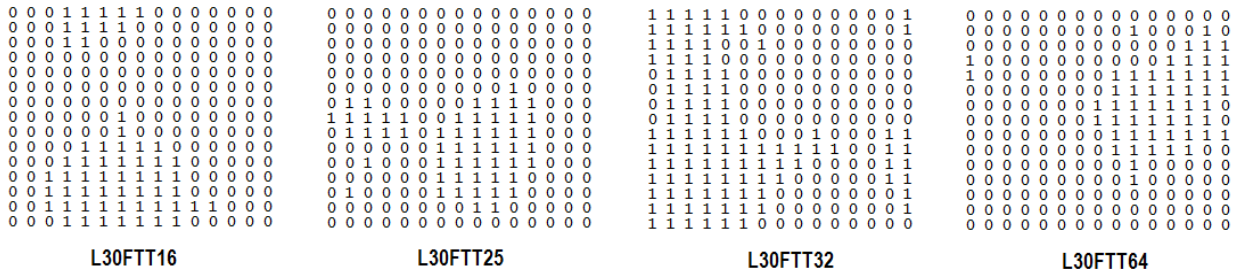

Figure 5 – seed: 2222, sample: 13

Conclusion

From our results, the ideal solution seems to be an overall smaller learning rate. We can also interpret that the learning rate and the its decay rate are correlated. I was able to get promising results (sample # 7 and 8) with an initial small learning rate paired up with a low or non-existent decay rate. Ideal results are possible with a greater learning rate, as long as its decay rate is relatively greater as well (sample # 9); this allows us to decay to a smaller learning rate as quick as possible. The differences between an overall greater and smaller learning rate can also be noted in Figure 1 and 2. The greater learning rate causes the noticeable displaced when compared to the results of a smaller learning rate. The greater learning rate can cause our clusters to fluctuate in position, allowing it to possible skip optimal placement.

The biggest factor that seems to improve our accuracy was the change in our Radius Decay Constant. Allowing the radius to decay at a faster rate concludes with small and concise clusters. The difference in between a small radius decay rate and a large radius decay rate can be noted in figure 1 and 3. The clusters are placed in the same position, however, the clusters formed in figure 3 are generally larger than the clusters form in figure 1. This would cause our results to be not generalized enough, therefore, leaving more room for error. Ideally, we can solve this by increasing the numbers of epochs, however, I stuck to a 1000 epochs because it was determined through empirical testing that epochs greater than a 1000 never provided significant improvements in accuracy.

When applying the Mexican Hat function to our experiment, it was noted that it was generally inferior to the Gaussian function. The results can be noted in Figure 4, when identical parameters were applied as sample # 1. The displacement caused by the Mexican Hat function to anything outside the radius causes the resulting clusters to be large and oddly shaped. We can apply the theory we mentioned above and reduce these clusters to by increasing the radius decay factor. Figure 5 displays the result of an increase radius decay rate; the results did manage to slightly improve but there is still a hint of oddly shaped clusters that can misrepresent our classification. In conclusion, Mexican hat function seems like that it can do well if applied to a specific problem.