ABSTRACT

Title of Thesis: FAULT DETECTION FRAMEWORK FOR IMBALANCED AND SPARSELY-LABELED DATA SETS USING SELF-ORGANIZING MAPS

Rushit N. Shah, Master of Science, 2018

Thesis Directed By: Professor Michael G. Pecht, Department of Mechanical Engineering

While machine learning techniques developed for fault detection usually assume that the classes in the training data are balanced, in real-world applications, this is seldom the case. These techniques also usually require labeled training data, obtaining which is a costly and time-consuming task. In this context, a data-driven framework is developed to detect faults in systems where the condition monitoring data is either imbalanced or consists of mostly unlabeled observations. To mitigate the problem of class imbalance, self-organizing maps (SOMs) are trained in a supervised manner, using the same map size for both classes of data, prior to performing classification. The optimal SOM size for balancing the classes in the data, the size of the neighborhood

function, and the learning rate, are determined by performing multiobjective optimization on SOM quality measures such as quantization error and information entropy; and performance measures such as training time and classification error. For training data sets which contain a majority of unlabeled observations, the transductive semi-supervised approach is used to label the neurons of an unsupervised SOM, before performing supervised SOM classification on the test data set. The developed framework is validated using artificial and real-world fault detection data sets.

FAULT DETECTION FRAMEWORK FOR IMBALANCED AND SPARSELY-LABELED DATA SETS USING SELF-ORGANIZING MAPS

by

Rushit N. Shah

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2018

Advisory Committee:
Professor Michael G. Pecht, Chair
Dr. Michael H. Azarian
Professor Peter Sandborn
Professor Patrick McCluskey

Dedication

To my parents Neeta & Nishad Shah for their continued support & encouragement.

Acknowledgements

First, I would like to thank my advisors Dr. Michael H. Azarian, and Prof. Michael G. Pecht for having provided me the opportunity to study and to conduct research at the University of Maryland, College Park, and at the Center for Advanced Life Cycle Engineering (CALCE). Their guidance and support throughout my two and a half years of education have helped me grow into a better researcher. I will forever be grateful to Dr. Azarian for the innumerable hours he spent discussing finer, technical aspects of projects with me, and challenging my thinking at every step along the way which has engrained in me the value of critical thinking.

Additionally, thank you to Prof. Patrick McCluskey and Prof. Hugh Bruck for serving on my committee and providing invaluable insights. I would like to extend a special thank you to the late Dr. Myeongsu Kang, without whose invaluable feedback this work would not have been possible.

I would also like to thank the many graduate students (Anto Peter, Jordan Jameson, Jing Tian, Guru Pandian, Varun Khemani, Nripendra Patel, Jose Romero and others) and research scientists at CALCE that I have had an opportunity to share my time with and learn from. Working with such a diverse and intelligent group has been an honor. Spending time with each of you in technical and, perhaps more importantly, non-

technical discussions provided me with support, guidance, or even simply a break which was instrumental to the completion of this work.

To my parents, who have fully supporting my choices and decisions and have helped me believe in all my dreams and aspirations, I am forever grateful. To my grandma, who unfortunately passed away a few months before the completion of this degree program, you would have been proud.

Finally, I'd like to thank the many companies that provide financial support to the research efforts at the University of Maryland and CALCE, making this work possible.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

With the rapid growth of technology, systems are more complex than ever before. High-value assets cost much more to operate and maintain than to manufacture and install. The difference in these costs is significant enough for businesses to adopt and to invest in preventative maintenance strategies [1]. Condition-based maintenance (CBM) involves monitoring the *health* of a system and scheduling maintenance activities accordingly [2]. Because system health is continuously monitored, organizations can reduce the cost of unplanned downtime and take maintenance action well before the occurrence of failure. When the monitored system is relatively simple and isolated, it may suffice to use a single sensor to monitor its health, even on a continuous basis. Vibration monitoring for bearings is an example of this kind of CBM. The system health can be monitored online, and any problem that may occur can be detected almost immediately. However, in the case of more complex systems, a single sensor is incapable of collecting adequate information to ascertain system health entirely. In such cases, multiple sensors are installed within the system and information from these different sensors is fused in an intelligent way to deduce system health and to detect incipient faults. This process is known as “fault detection”.

Fault detection approaches can be classified into two main categories viz. model-based, and data-driven. The model-based approach exploits knowledge of the mathematical representation of the system to detect incipient faults. This approach, however, requires

incorporating domain knowledge and a physical understanding of the target system [3] [4]. However, one does not always have such pre-requisite knowledge at one's disposal when developing a fault detection methodology for a target system. In some cases this may be due to the manufacturer's reluctance to share system models due to competitive reasons. It may also be because due to difficulty in explicitly modeling complex systems such as aircraft where the behavior of the system is an aggregation of hundreds of interdependent components. However, in such cases system monitoring data, collected from a variety of sensors installed on the system, may be available and can be used to develop data-driven fault detection methodologies. In the domain of data-driven fault detection methodologies, machine learning techniques have been widely employed in recent years. Some examples of machine learning techniques for fault detection include support vector machine (SVM) and its variants in rotating machinery, pumps, and HVAC systems [5] [6]; Bayesian inference-based Gaussian mixture models (GMMs) for non-Gaussian processes [7] [8]; artificial neural networks (ANNs) for transmission lines [9]; and deep neural networks (DNNs) [10]- [11] for drivetrains, microgrids and propellant loading systems. The previous examples do not imply that these techniques have only been applied to the application domains mentioned here—the application of these machine learning techniques is widespread across all domains of fault detection. A survey of machine learning techniques studied in mechanical systems research is provided in [12].

An imbalanced data set is one where one or more classes of data (called the majority classes) are represented by a much larger number of observations in the training data set than the other classes (called the minority classes). Most of the aforementioned machine learning techniques in [5] [6] [7] [8] [9] [10] [11] [12] function under the assumption that the data used to train the model is balanced. In the absence of balanced data sets, these techniques provide suboptimal results [13]. The learning process of these classifiers involves adjusting model parameters such that performance metrics, such as the prediction accuracy on the training data set, are minimized. The imbalance problem induces a bias in the model towards the majority class, while patterns in the minority class do not get learnt, even though the model yields a high overall accuracy [14] [15]. Classifiers which do not account for the class imbalance in the training data set can achieve a high prediction accuracy by learning incorrect decision rules which carry no practical value [14]. As an example, assume a training data set contains 990 observations from the positive class, and 10 observations from the negative class. A binary classifier trained on such a data set would achieve a 99% prediction accuracy by merely predicting every input observation to belong to the positive class. However, such a classifier would yield ~50% accuracy on a test data set which contains an equal number of observations from both the positive and negative classes, effectively rendering it useless for any practical deployment. Minority instances being incorrectly treated as noise and vice-versa [16]; low density of observations; and small sample size with high dimensional data [17], are some of the other challenges faced by traditional classifiers in the presence of imbalanced data sets. Fault detection frameworks developed using traditional machine learning classifiers are also prone to such issues since imbalanced data sets are encountered frequently in this application domain. This is because any system is expected to mostly operate in its healthy state, and consequently the volume of condition monitoring data collected from the healthy state of the system is larger than that collected from the system operating in a faulty state. Thus, when a machine learning model is trained, the condition monitoring training data set is inherently imbalanced [18]. In such cases, a model's inability to account for this imbalance while training to predict future occurrences of the fault pose a risk to not only the financial standing of an organization, but also human life. In other words, the rare events comprising the

minority class of data occur infrequently, but the cost of misclassifying them can be heavy. For example, the inability to detect a rare fault in a subsystem in a manufacturing plant can have catastrophic consequences such as the propagation of damage to other systems in the plant (if systems are interdependent), to more financial consequences such as unplanned downtime.

Another assumption which is central to the success of the methodologies in [5] [6] [7] [8] [9] [10] [11] [12] for detecting faults is the availability of labeled training data. These approaches constitute a class of machine learning methods based on supervised learning, where sets of observations representing both healthy and faulty modes of operation of the system are available. However, condition monitoring data, in its raw form, is unlabeled. Labeling this data for use in supervised fault detection methodologies requires manual annotation of observations by subject matter experts. The task of manually labeling observations by these experts is time-consuming and expensive [19] [20]. In *situations* where the amount of labeled observations available is limited but unlabeled observations are available in abundance, semi-supervised learning techniques provide an economical alternative to supervised learning techniques for fault detection. Semi-supervised learning techniques are a class of machine learning techniques which utilize both labeled, and unlabeled samples to perform classification. These techniques aim to extract information associated with healthy and faulty system conditions which is contained in the spatial structure of unlabeled observations in the condition monitoring data [21].

In light of these issues, this study proposes a methodology based on self-organizing maps (SOMs), which can be used to detect faults in scenarios where the available condition monitoring data is either imbalanced or has only a few observations labeled. To deal with imbalanced data sets, SOMs are trained in a supervised manner on every available class of data, before performing classification using the cSOM approach [22]. The SOMs trained on each class of data are chosen to have the same map size, such that the neurons of the trained SOMs represent a balanced training data set. This map size of the SOMs is determined by solving a multiobjective optimization problem to minimize the information loss; the training time; and the prediction accuracy on a test set. For data sets containing mostly unlabeled samples, SOM-based, semi-supervised approach is adopted. First, SOM is used as a clustering technique to capture the patterns in the training data of both, labeled and unlabeled samples. Then, using the labeled samples as training data, and the neurons of the trained SOMs as test data, the SOM neurons are assigned class labels. This process of transductive learning is also referred to as pseudo-labeling. Note, the unlabeled samples are only used to capture the distribution of data, and are not labeled. Then, the labeled SOM neurons are used for classification using the cSOM.

The rest of the study is divided as follows. Chapter 2 discusses work done previously on imbalanced and sparsely-labeled data sets in the context of fault detection. Chapter 3 is dedicated to providing a brief overview of SOMs. Chapter 4 details the supervised SOM-based fault detection methodology, cSOM, developed in [22]. In Chapter 5 the

cSOM methodology is developed for alleviating the imbalance problem. Hyperparameter considerations for the cSOM are discussed, and a Genetic Algorithm (GA) –based multiobjective optimization strategy is discussed to achieve class balance by minimizing cSOM's fault prediction accuracy, training time, and other SOM quality metrics. Experiments are conducted using artificial data and results presented along with comparisons with other classifiers. In Chapter 6 the cSOM methodology is developed for fault detection in cases where training data sets contain mostly unlabeled observations. Issues related to sparsely-labeled data are discussed, the developed methodology is explained, and experimental results are presented along with a comparison with other classifiers.

Chapter 2: Related Work

This chapter discusses work done by other authors on the issues of imbalanced and sparsely-labeled data sets in the context of fault detection. Previously proposed techniques, and their benefits and drawbacks are discussed. How our developed methodology helps overcome those drawbacks is also pointed out.

2.1. Imbalanced Data Sets

Recently, approaches to fault detection have been proposed to deal with imbalanced condition monitoring training data sets. These approaches can be broadly divided into two categories – classifier-based, and preprocessing approaches. Among classifier-based techniques, a technique which integrates one-class support vector machines and binary tree was introduced to deal with imbalanced data sets in [23]. The separability measure used to construct the binary tree in this study was Mahalanobis distance. However, the performance of this technique, was shown to be heavily dependent on the specific separability measure used to build the binary tree. Also, training one-class SVMs on underrepresented classes of data leads to severe overfitting. Another fault detection technique based on Extreme Learning Machine (ELM) was presented in [24]. The strategy was developed to reduce the information loss incurred when dealing with imbalanced data sets. However, the addition of new, previously unseen classes of data required training a large number of classifiers to be trained on the new data, which added to the complexity of the ensemble used for classification [25].

Preprocessing techniques are more widely-used to deal with imbalanced data sets. The approaches include feature selection and extraction methods, and resampling methods. Among feature selection/construction methods, methodologies based on the unsupervised manifold learning methods were employed in [26] [27]. Firstly, by using the unsupervised manifold learning techniques, these methods failed to fully utilize the value of the labeled data available. Secondly, since the techniques in [26] [27] were based on extraction of time- and frequency-domain signals from vibration data collected from rotating machinery, their application was found not to be generalizable to other types of complex systems where it may not be possible to characterize the behavior of components or subassemblies, as it is in the case of rotating machinery where fault frequencies are deterministic. A resampling technique called Similarity-based Undersampling and Normal Distribution-based Oversampling (SUNDO), which combined oversampling the minority class and undersampling the majority class, was introduced in [28]. Although the study has been widely cited, the performance of the technique, as claimed, could not be validated since the computation of classification accuracy was incorrect across experimental results. Also, the method is restricted to binary classification problems, and thus cannot be utilized to detect multiple faults, or new, previously unseen faults that may arise in a system.

The methodology developed in this study aims to account for the imbalance in the training data set using our previously-developed supervised SOM methodology. In this methodology, a SOM is trained, individually, on each class of observations in the

training data set. At the end of the training, the neurons of each SOM are prototypes of the observations of the class of data on which the SOM, to which the neurons belong, was trained. Classification is then performed by computing the relative distance of an unlabeled test observation to the closest neuron in each class of data. Balance is achieved by using the same number of neurons to train each class-specific SOM. Since only the SOM neurons are used for classification, their equal number ensures a balanced data set, while preserving the patterns in the original training data. The SOM map size to be used for the class-specific SOMs is determined by solving a multiobjective optimization problem. The size of the class-specific SOMs, and the size of the neighborhood function are used as the independent variables. The difference in the information entropy of a trained SOM and the class of data on which it was trained; the training time; and the classification error on a test data set; are used as objective functions to be minimized for this optimization problem. The solution to the optimization problem yields a map size which balances the training data set; minimizes the information loss during SOM training; minimizes the training time for computational efficiency; and yields the lowest classification error. Like the approach in [28], our method involves both, oversampling the majority class, and undersampling the minority class. However, unlike their approach, our developed method is not restricted to binary fault detection problems. In other words, our method can not only be used to determine if a system is in the healthy or faulty state, but can also be used to determine which specific type of fault has occurred, provided there is at least some training data available associated with all healthy and faulty states of the system.

2.2. Sparsely-Labeled Data Sets

While the issue of exploiting sparsely-labeled training data sets to build classifiers has been studied widely in general machine learning literature [29]- [30], it has received less attention in the context of fault detection applications [31]. A Gaussian Mixture Model (GMM)-based semi-supervised learning technique was introduced in [32]. In this study GMM was used to cluster the labeled and unlabeled data. A semi-supervised coefficient β was introduced to weight observations during while estimating clustering parameters, to account for most observations being unlabeled. The unlabeled samples were probabilistically classified into different classes during the parameter update step of the clustering. While novel in its approach, the method was shown to suffer from two major drawbacks. Since the clustering method chosen was parametric, a drift in the mean and covariance of the incoming unlabeled data or future test data could lead to deterioration in performance. More importantly, the use of GMM as the clustering algorithm restricted the application of this technique to near-Gaussian distributions of healthy and faulty data [33]. SOM is a clustering technique which is not restricted by the underlying distribution of the training data. It can achieve good clustering results with simple Gaussian data, and highly nonlinear, non-Gaussian data, too. SOM is a type of artificial neural network whose weights are adjusted during training in a manner such that the neurons develop into prototypes of the training observations. However, in its native form, SOM is an unsupervised learning technique. The method proposed in [34] was specifically for prediction of faults in software. However, its application was not found to be restricted to just software fault prediction. The authors used SOM due

to its advantages over other clustering algorithms. A hybrid SOM for semi-supervised learning was developed. The method involved first training the SOM in an unsupervised fashion using the labeled and unlabeled training data. 'Dead neurons' which were never activated during the training procedure were then eliminated to reduce noise. The neurons of the SOM, which, after training, were representative of the training data were assigned class labels. The labeling was performed by setting pre-determined thresholds on the individual features of the neurons. For example, if the training data consisted of D-dimensional observations, D threshold values were obtained from previous studies and expert knowledge. The neurons of the SOM, which retain the dimensionality of the training data, were then compared with these pre-determined thresholds. If the values of more than $D/2$ features of a neuron was found to exceed their respective thresholds, the neuron was labeled faulty. The labeled SOM neurons, and their respective labels, were used as inputs to train a multilayer perceptron (artificial neural network), which could then be used as a model for predicting the health state of future test observations. The methodology, although simple in its formulation, was found to have multiple drawbacks. Firstly, the threshold required to label SOM neurons as into different classes must be pre-determined which, without exception, requires expert knowledge. In cases where thresholds may already be available for one type of system, the model thus developed would be applicable only to that system, leading to poor generalization. Also, the use of thresholds to classify neurons, and not the labeled training observations, completely bypasses the latter. The authors failed to take advantage of the class label information available from labeled

observations. The hybrid SOM approach developed in [34] was found to be more unsupervised than semi-supervised since the training observations were used only for clustering.

In this study, the methodology developed to deal with sparsely-labeled condition monitoring data sets is similar to the hybrid SOM approach in [34]. A SOM is first trained in an unsupervised manner using both labeled and unlabeled observations, for clustering. Once training is complete and the SOM neurons resemble prototypes of the training observations, the labeled observations are used as the training data to train a 1-nearest neighbor (1NN) classifier [35]. The trained neurons are used as test observations and assigned labels using the trained 1NN classifier. While the authors in [34] used the labeled SOM neurons to train an artificial neural network for classification, in this study, the supervised cSOM approach for classification using the labeled neurons is used. The developed approach proves to be significantly cheaper computationally, since no further training is required. Future test observations are classified merely based on their relative distance to the nearest-neuron in each class.

Chapter 3: Self-Organizing Maps

3.1. Overview

A self-organizing map (SOM), also known as a Kohonen neural network, is a type of unsupervised machine learning technique based on competitive, instance-based learning [36]. In its native form, the SOM creates a neural network that retains information associated with the topological relationships within the training data set. A SOM is composed of several artificial neurons, each associated with a weight vector of the same dimension as that of the training data set. The neurons are grouped based on the similarity of their weights such that neurons with similar weights are neighbors. The topological relationships in the training data set are reflected in the neighborhood relationships of the neurons.

To create a SOM, the input data is first normalized by calculating the z-score of each observation on a per-variable basis. There is no theoretical basis for determining the size of the SOM, and it varies depending on the input data set. The size is thus empirically determined by calculating the number of neurons as

$$M \approx 5\sqrt{n} \qquad (1)$$

where M is the number of neurons and n is the number of observations in the training data set. The neurons are organized on a two-dimensional map such that the ratio of the

lengths of sides of the maps is approximately equal to the ratio of the two largest eigenvectors of the covariance matrix of the training data set. There is no consensus on a single initialization strategy for SOM neurons that provides optimal performance. Valova et al. [37] found that initializing the neurons on a self-similar curve such as Hilbert, provided the satisfactory coverage of the topology of the training data set. Similarly, random initialization has provided satisfactory performance. Accordingly, this study employs a random initialization strategy due to its simplicity. The weights of all neurons w_{ij} are initialized to random numbers in the range $(0,1)$, i, j are the row and column of the SOM lattice, respectively. In a SOM, a measure of similarity can be defined as the Euclidean distance *d* between the input vector *x* and the weight vector *w* of the given neuron, which is computed as follows:

$$d(x, w) = \|x(t) - w(t)\| \tag{2}$$

where t is the number of a current iteration. Then, the neuron on the map that is closest to the input vector in the Euclidean space is referred to as the best matching unit (BMU), also known as a winning neuron. That is, the BMU is the neuron with the weight such that:

$$BMU = \underset{i,j}{\operatorname{argmin}} \left\| x(t) - w_{ij}(t) \right\| \tag{3}$$

During the training process, the neuron's weights are updated to increase the similarity with the input neuron:

$$w_{ij}(t+1) = w_{ij}(t) + h_{ij}^{c}(t)[x(t) - w_{ij}(t)] \qquad (4)$$

where $h_{ij}^{c}(t)$ is the neighborhood function for the BMU c. In fact, a SOM introduces the neighborhood function to preserve the topological properties of the input space. The neighborhood function depends on the lattice distance between the BMU (neuron c) and the other neurons on the map. In the simplest form, it is 1 for all neurons close enough to the BMU and 0 for others. However, a Gaussian function is a common choice for the neighborhood function, defined as:

$$h_{ij}^{c}(t) = a(t)e^{\left(\frac{\|w_c(t) - w_{ij}(t)\|^2}{2\sigma^2(t)}\right)} \qquad (5)$$

where $\alpha(t) = \alpha(0)\frac{1}{t}$ is a learning rate at iteration t, $\alpha(0)$ is an initial learning rate, $w_c(t)$ is the weight vector associated with the BMU c, $w_{ij}(t)$ are the weight vectors of the neurons on the map, σ is the radius around the BMU c. According to (5), the neighborhood function shrinks as the iteration increases. At the beginning when the neighborhood is broad, the SOM takes place on a global scale. When the neighborhood has shrunk to a couple of neurons, the weights are converging to local estimates. The SOM is trained iteratively until all neurons are grouped into clusters.

3.2.Use of SOM for Fault Detection

As described in the previous subsection, SOM, in its traditional form, is an unsupervised learning technique. It can learn the structure of the training data without considering the labels of the training observations. However, even in its unsupervised form, SOMs have successfully been implemented in fault detection applications. [38] [39] [40] [41] [42] [43]. Fault detection was performed on a cooling fan in [44], and a wireless sensor network in [45]by utilizing the distance of a test observation from the closest-neuron of a SOM trained on only the healthy class of training data. SOMs were implemented for detecting process faults in [46] [47] by operators visually inspecting whether a healthy or a faulty region of the SOM was activated by a test observation. In [48] a shallow neural network was used to estimate the remaining useful life (RUL) of bearings, using the distance of test observations from the closest-neuron as the input to the network.

There are multiple benefits of using SOMs. Firstly, since SOM is a non-parametric learning technique, it does not make any assumption about the underlying distribution of the training data. This allows SOMs to be used for clustering nonlinear data sets, unlike methods like GMMs. Secondly, SOM can be used to generate a specified number of prototypes of the training observations. Additionally, it is possible to modify the unsupervised learning procedure of SOM to one of supervised nature. These benefits, together, can be used to harness the power of SOMs to alleviate the problems of imbalanced and sparsely-labeled data sets in fault detection.

Chapter 4: cSOM – A Supervised Fault Detection Methodology

This chapter first provides an illustrative overview of the developed fault detection methodology. Then, the method is formulated as a general binary classifier followed by its formulation as a generalized classifier for multiclass data sets. Experimental results, using artificial and benchmark data sets, are then presented for both the two-class and multi-class classifier.

4.1. Overview of cSOM

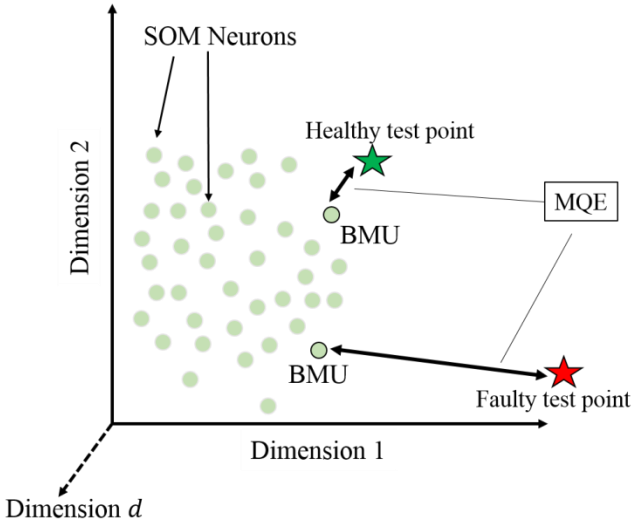

Figure 1: The MQE-based fault detection concept, where d represents the dimensionality of the input vector of the data set and is the same as that of the weight vector of the SOM neurons.

Conventionally, the minimum quantization error (MQE) has been used to indicate whether a target system or component is in a faulty/healthy state [49]. More

specifically, the MQE is defined as the distance between the input vector of a test observation and the weight vector of the BMUs in the trained SOM:

$$MQE = \min_k \|v - w_{BMU_k}\| \quad (6)$$

where v is the input vector and w_{BMU_k} is the weight vector of the k^{th} BMU. For fault detection, the SOM is first trained using only healthy observations. As test observations are collected from a system or component being monitored, the MQE is calculated for each of those test observations. The MQE-based fault detection method operates on the hypothesis that a large value of the MQE indicates that the associated test observation belongs to a part of the Euclidean space that is not represented by the training data, i.e., the test observation belongs to the fault class. This idea is illustrated in Figure 1. Based on the assumption that any deviation from the space covered by the healthy training data is regarded as the system being faulty, the MQE can be used to indicate the severity of the system's deviation from normal.

Unlike the conventional use of the SOM in fault detection, the developed cSOM method employs multiple SOMs; one SOM is trained individually, on observations from each class of data. A data set consisting of a healthy and faulty class would thus have two SOMs – one trained on the healthy data, and the other on faulty data. For such a data set, at the end of the training phase, the developed method outputs two sets of neurons; one represents the healthy class, whereas the other represents the faulty class.

This is illustrated in Figure 2. This is the reason the method is referred to as cSOM. For a c-class problem, the method requires training c different SOMs, one on each class of data. Thus, for a two class problem, the algorithm is referred to as 2SOM.

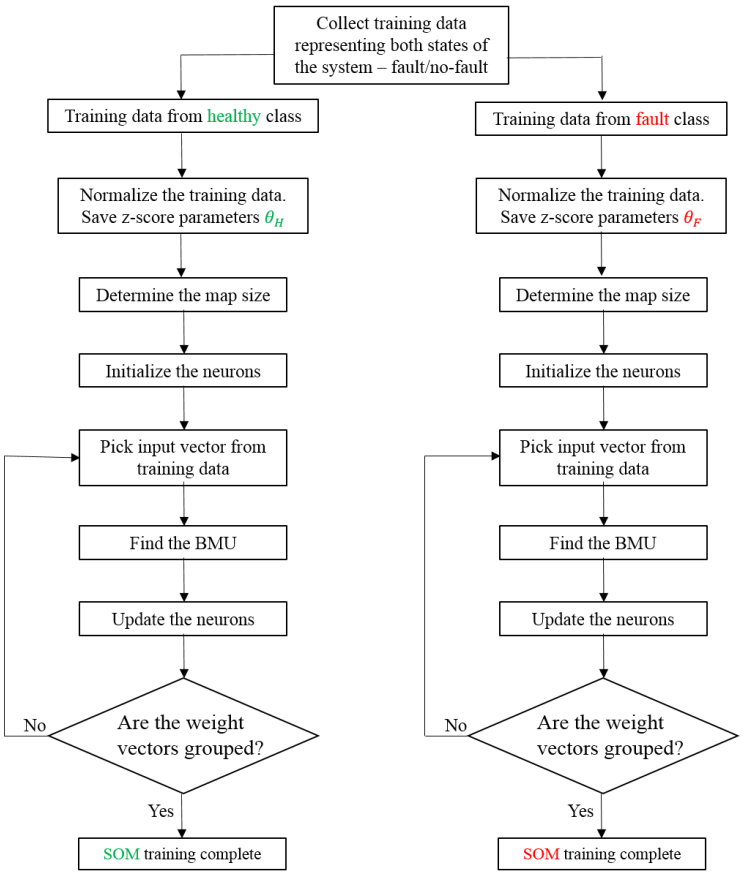

Figure 2: The training procedure of the cSOM methodology.

Further, the developed method stores the z-score parameters θ_{H} and θ_{F}, where θ_{H} and θ_{F} include the mean and standard deviation of the healthy and faulty training data set, respectively, and are used later to preprocess test observations. After the training phase is complete, the training data sets can essentially be discarded since the trained SOM neurons adequately represent the patterns in them. Unlike the conventional fault

detection approach using the MQE between a test observation and trained BMUs, the developed method introduces a variant of the MQE-based metric that represents the degree of healthiness, i.e., $D_{\text{healthiness}}$, using a ratio of the healthy MQE (d_{h} in Figure 3) to the faulty MQE (d_{f} in Figure 3) for a given test observation.

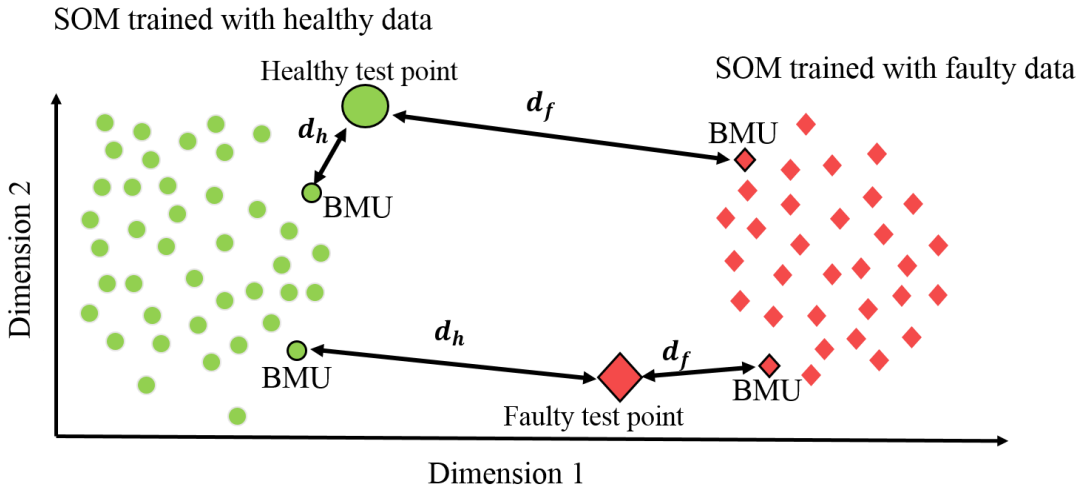

Figure 3: An example of decision-making of a test observation using $D_{\text{healthiness}}$.

Accordingly, $D_{\text{healthiness}}$ is used to determine which of the two states (i.e., faulty or healthy) of the system it represents, as illustrated in Fig. 4. The two terms MQE_{healthy} and MQE_{faulty} must be combined in a logical way so as to describe the state of the given test observation. As briefly mentioned above, these terms are used to compute a new metric $D_{\text{healthiness}}$ for making a decision on the given test observation. The metric is the normalized ratio of the two MQEs. Although simple, the metric provides a good handle on the similarity of the test observation to both the faulty and healthy states of the system. It is mathematically formulated as:

$$D_{\text{healthiness}} = \frac{MQE_{\text{healthy}}}{MQE_{\text{healthy}} + MQE_{\text{faulty}}} \qquad (7)$$

As illustrated in Figure 3, for a test observation that is collected from a healthy state of the system, a small value of MQE_{healthy} may be expected owing to its similarity to that set of BMUs. In contrast, a much larger value of MQE_{faulty} may be expected from such a test observation. Thus, computing $D_{\text{healthiness}}$ for such a test observation would yield a number that is close to zero. Conversely, for a test observation collected from a faulty state of the system, larger values of MQE_{healthy} and smaller values of MQE_{faulty} may be expected, which consequently yield a value closer to one for such a test observation. Thus, informally stated, $D_{\text{healthiness}}$ is a metric of the "*healthiness*" of the system. The $D_{\text{healthiness}}$ value can also be thought of as the probability that the system is in a healthy state, given that the test observation was recorded:

$$P(fault\ state|T) \cong D_{\text{healthiness}} \qquad (8)$$

where T is the test observation. This metric may be used for fault detection by setting a threshold on $D_{\text{healthiness}}$. Since the metric is a normalized ratio of MQEs, a simple threshold of 0.5 may be considered. That is, a $D_{\text{healthiness}}$ less than 0.5 would imply that the test observation is closer to the healthy class than to the faulty class.

4.2. Formulating a Binary Classifier

The generalized procedure of the developed supervised fault detection method can be summarized as follows. Given a set of training observations

$$(\bar{x}_i, y_i) \; \forall \; i = 1, \dots, n \tag{9}$$

$$\bar{x}_i \in \mathbb{R}^d \tag{10}$$

$$y_i \in \{\omega_1, \omega_2\} \tag{11}$$

where n is the number of training observations, $\bar{x}_i$ are the d-dimensional training observations, and y_i is the label of the training vector $\bar{x}_i$, and ω_1 and ω_2 are the labels for the two classes of data. One SOM is trained on all $\bar{x}_i \in \omega_1$, and another SOM on all $\bar{x}_i \in \omega_2$. The training procedure is as illustrated in Figure 2. The training yields two sets of SOM neurons with weight vectors

$$\overline{w_j^1}, \overline{w_j^2} \in \mathbb{R}^d \; \forall \; j = 1, \dots, m \tag{12}$$

where m is the number of neurons used for training each SOM, $\overline{w_j^1}$ are the weight vectors of the neurons associated with class ω_1, and $\overline{w_j^2}$ are the weight vectors of the neurons associated with class ω_2. After training, given a set of test observations

$$\bar{t}_i \; \forall \; i = 1, \dots, z \tag{13}$$

$$\bar{t}_i \in \mathbb{R}^d \tag{14}$$

where z is the number of test observations, compute the MQE of each test observation from both SOMs as

$$MQE_{\omega_1} = \min_j \left\| \overline{t_\iota} - \overline{w_j^1} \right\| \quad (15)$$

$$MQE_{\omega_2} = \min_j \left\| \overline{t_\iota} - \overline{w_j^2} \right\| \quad (16)$$

where $MQE_{\omega 1}$ is the MQE to the SOM representing class ω_1 (e.g., healthy class), and $MQE_{\omega 2}$ is the MQE to the SOM representing class ω_2 (e.g., faulty class). Then, the probability that the given test observation belongs to either ω_1 or ω_2 can be computed as follows:

$$P_i(\omega_1|\overline{t_\iota}) = \frac{MQE_{\omega_1}}{MQE_{\omega_1} + MQE_{\omega_2}} \quad (17)$$

$$P_i(\omega_2|\overline{t_\iota}) = \frac{MQE_{\omega_2}}{MQE_{\omega_1} + MQE_{\omega_2}} \quad (18)$$

Then, a decision rule can be formulated as:

$$P_i \leq 0.5 \; \forall \; \overline{t_\iota} \in \omega_1 \quad (19)$$

$$P_i > 0.5 \; \forall \; \overline{t_\iota} \in \omega_2 \quad (20)$$

The threshold of 0.5 is the simplest case for this classifier. Geometrically, this classification rule may be interpreted as follows – the test observation is assigned class label ω_1 if it lies less than halfway between its BMU in either SOM, and class label ω_2 otherwise.

4.3. Formulating a Multiclass Classifier

The metric derived in the previous subsection can be obtained under simplifying assumptions for a more general case. In the case of multiclass data sets, a SOM must be trained on each class of data available. Thus, for a c-class problem, c different SOMs must be trained. The generalized metric can then be formulated as

$$P_i(\omega_j|\bar{t_\iota}) = 1 - \frac{MQE_{\omega_j}}{\sum_{j=1}^{c} MQE_{\omega_j}} \tag{21}$$

where ω_j is class j, $j \in \{1, \dots, c\}$; c is the number of classes; and MQE_{ω_j} is the MQE from the j^{th} class. The class for the test point t_i is predicted as

$$\omega_i = \underset{j}{\operatorname{argmax}}\, P_i(\omega_j|\bar{t_\iota}) \tag{22}$$

where ω_i is the predicted class of the test observation. For the case of binary classification (i.e., c = 2), the metric in equation 21 is the same as the metrics in equations 17 and 18.

4.4. Binary Classification Experimental Results

Experiments were performed using real-world and artificially generated data sets. While the real-world data sets were benchmarked and thus allowed the comparison with other classifiers to be benchmarked too, the artificial data sets allowed us to include specific artifacts within the data and then compare the response of various

classifiers to those artifacts. Based on results from these experiments, the developed method and kernel SVM technique are compared in terms of their operation. It is shown how the two methods can yield comparable results and how their computational costs vary with increasing size of the training data set. The performance of the developed method is compared with that of the kernel SVM technique.

4.4.1. Experiments Using Benchmark Data Sets

An experimental study was conducted to gauge the relative performance of the developed supervised classifier against other widely used ones. The classifiers used for comparison include linear discriminant analysis (LDA) classifiers, quadratic discriminant analysis (QDA) classifiers, k-nearest neighbors (k-NNs), a kernel version of SVM using the Gaussian radial basis function (SVM-RBF), and decision trees (DTs), and neutral networks (NNs). These classifiers were chosen to add diversity to the study in terms of how they find the decision boundary between two classes.

The data used for this study was obtained from the publicly available UCI Machine Learning Repository [50]. The data sets used for this study were Iris, Wine, Breast Cancer Wisconsin (BCW), and Climate Model Simulation Crashes (CMSC). These data sets were chosen because of their varying levels of data complexity. Iris is one of the most popular data sets in pattern recognition literature and contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The wine data set is the result of a chemical analysis of wines grown in the same region in Italy, derived from

3 different cultivars. The analysis determined the quantities of 13 constituents found in each of the 3 types of wines. The CMSC data set contains records of simulation crashes encountered when quantifying the uncertainties of 18 model parameters within the Parallel Ocean Program (POP2) component of the Community Climate System Model (CCSM4). Out of the 540 simulations, 46 failed for numerical reasons at certain combinations of parameter values. The objective of this data set is to use classification to predict simulation outcomes (fail or succeed) from input parameter values. The Breast Cancer Wisconsin (BCW) data set contains features that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. Each observation in this data set may be classified as either 'malignant' or 'benign'. Table 1 provides some relevant details about each of these data sets. For data sets containing more than two classes of data, two classes were randomly chosen to evaluate performance since the classifier developed so far is binary and thus compatible with only two classes of data.

Table 1: Details of benchmark data sets used for experimental study.

Data Set	**Number of Features**	**Number of Observations**
Iris	4	150
Wine	13	178
CMSC	18	540
BCW	32	569

For each data set, 1000 observations were randomly sampled from each class and used for training each classifier. Owing to the fact that not all data sets contain an adequate number of observations, the training data set was obtained as a result of uniform oversampling. Similarly, 250 observations were randomly sampled from each class and

used for testing. The performance of each classifier was reported as the percentage classification accuracy of that classifier on the test set of each of the four data sets.

Table 2: Summary of classification results on benchmark data sets, where the top three classifiers for each data set are highlighted in bold.

Classifier	**Iris (%)**	**Wine (%)**	**CMSC (%)**	**BCW (%)**
2SOM	**100.00**	**99.80**	**99.60**	**96.80**
LDA	**100.00**	99.60	96.80	89.40
QDA	**100.00**	99.20	97.60	92.80
2NN	**100.00**	99.40	97.80	90.40
5NN	**100.00**	99.20	**98.00**	88.80
SVM-RBF	**100.00**	**100.00**	**98.00**	**97.80**
DT	**100.00**	**100.00**	97.40	**96.20**
NN	**99.80**	**99.80**	**99.80**	**99.60**

summarizes the results of this study. The Iris data set is known to be linearly separable, and thus even a simple linear classifier such as LDA provided 100% test accuracy. In two of the other three data sets (Wine and BCW), SVM-RBM yielded better performance than 2SOM and the six other classifiers being compared. In the case of the CMSC data set, the developed classifier outperformed other classifiers. The performance of the developed classifier was consistently found to be close to that of the SVM-RBF. It was encouraging to find that despite the developed classifier being

similar in its operation to the nearest neighbors, it performed significantly better than both 2-NN and 5-NN. This observation is discussed in detail in Chapter VIII.

4.4.2. Experiments Using Artificial Data Sets

It is known that the SOM training procedure can capture nonlinearities in the data. This is also evident in the performance of the developed classifier on the more complex data sets, such as CMSC and BCW in the previous section. However, to understand which type of nonlinearities the developed classifier can handle, and how well it performs the task of classification in their presence, a simulation study was conducted. Two-dimensional artificial data was used for this study for easy visualization. The data was seeded with complexities that are known to be detrimental to classification results. The following four cases of increasing complexity were used to generate artificial two-class data sets: unimodal Gaussian data with overlap (U-GAUSS), multimodal Gaussian data (M-GAUSS), nonlinear data with spiral classes (2SPIRAL), and multimodal nonlinear data with spiral classes (M-2SPIRAL). These artificially generated data sets are visualized in Figure 4. The unimodal Gaussian case is representative of noise in the training data. The overlap region renders classification difficult for any classifier. The other three cases are meant to represent various multimodalities and nonlinearities in the data that may arise.

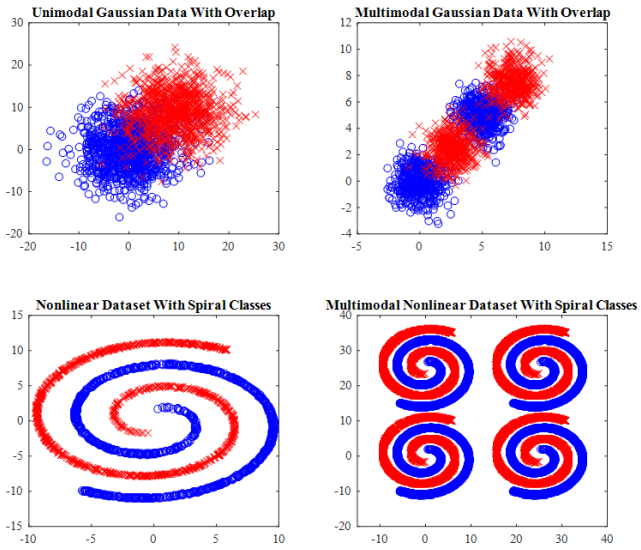

Figure 4: Visualization of the two-dimensional artificial data sets generated to test the sensitivity of the developed classifier, the two classes are denoted with ‘o’ and ‘x’.

The methodology used to test the performance of the developed classifier in this study was the same as that of the experimental study with benchmark data sets. For each data set, 1000 observations and 250 observations were sampled from both classes for training and testing, respectively. The test performance of the developed classifier and that of the six other classifiers used in the previous section in each case are presented in Table 3. In the seemingly simple case of unimodal Gaussian class distributions with overlap, the performance of the developed classifier suffered the most. This result was expected because overlapping data proves challenging for most classification algorithms, with the possible exception of kernel methods which operate in potentially infinite-dimensional space. In the region of overlap between the two classes of data, several neurons from both classes may coexist. Thus, within that dense region of

neurons, it is more likely that a test observation that was actually obtained from one class is closer to the BMU of the other, thus causing it to be misclassified. The performance of classifiers generating linear and quadratic decision boundaries (LDA and QDA) suffered when bimodalities and nonlinearities were introduced in the data, as was expected from such classifiers. In every case, SVM-RBF yielded classification accuracies greater than ~89%, thus establishing its value as a consistently reliable classifier on even complex data sets.

Table 3: Comparison of percentage classification accuracy of 2SOM and other classifiers, on the artificially generated data sets.

Classifier	**U-GAUSS (%)**	**M-GAUSS (%)**	**2SPIRAL (%)**	**M-2SPIRAL (%)**
2SOM	89.20	92.20	97.80	90.80
LDA	90.20	51.60	68.00	56.80
QDA	89.60	52.20	68.00	56.40
2NN	86.80	91.40	97.80	90.40
5NN	87.20	93.60	98.00	88.80
SVM-RBF	89.00	93.00	98.00	97.80
DT	89.00	92.20	97.40	96.20

4.4.3. Comparison of 2SOM and SVM-RBF

The results for the benchmark and artificial data sets provided valuable insight into the operation of the 2SOM classifier. The algorithm classified test observations based on their relative distance to the nearest prototype in each class of training data by

computing the normalized ratio of the minimum distance from every class of data. Since the first nearest-neighbor distance was used to compute the ratio, one might expect that this algorithm would yield performance close to that of a kNN classifier, specifically 1NN. However, the fundamental difference between 2SOM and the nearest-neighbor family of algorithms is that while the nearest-neighbor algorithm chooses the k closest neighbors to a test point regardless of their class labels, 2SOM picks the nearest neighbor from *each* class, individually. In fact, 2SOM's performance was found to match the performance of kernel SVM-RBF more closely than that of k-NN. By choosing a nearest neighbor from each class of data for a given test observation, 2SOM forces those BMUs to essentially behave as support vectors for that test observation. Indeed, this interpretation of 2SOM is in agreement with the geometric convex hull interpretation of SVM [51]. Herein lies the similarity between 2SOM and SVM. To investigate the differences between the two algorithms, another simulation study was conducted. While the performance of the two algorithms on the artificial data sets used so far was comparable, the size of the training data had not been taken into account. The M-2SPIRAL data set used in the previous section was also used for this study. The training time for both 2SOM and SVM-RBF was recorded with an increasing number of observations used for training. The size of the training set was varied from 2,000,000 observations to 180,000,000 observations. The test accuracy was recorded on a test set of 1000 observations, which consisted of observations not present in the training data. Figure 5 and Figure 6 show the results from this study. The

test accuracy of SVM-RBF was found to be consistently between ~96–100%, whereas that of 2SOM varied between ~90–96%.

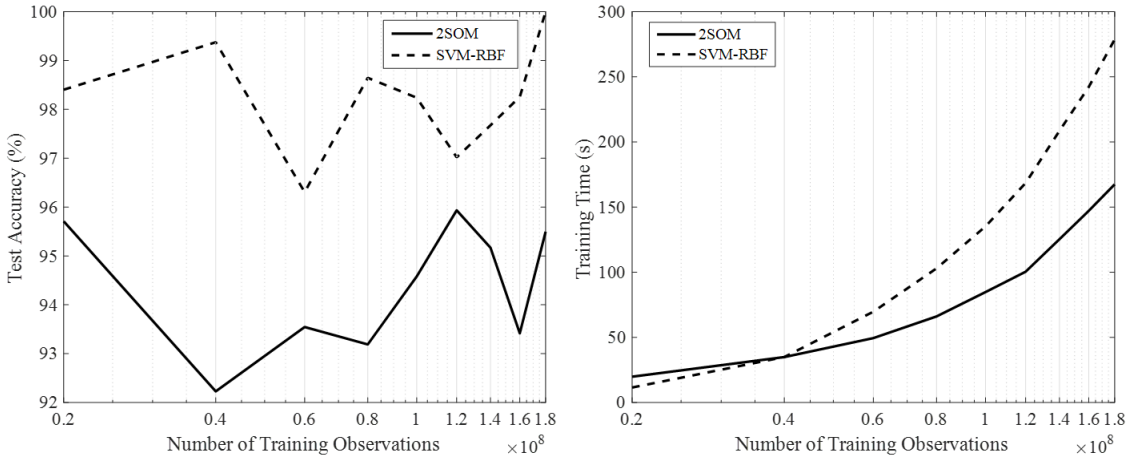

Figure 5: Comparison of (a) test accuracy, (b) training time for 2SOM and SVM-RBF classifiers as functions of increasing size of training data set (M-2SPIRAL).

However, the training time of the 2SOM was found to be significantly lower than that of SVM-RBF as the size of the training data set increased. In the case of the largest data sets, 2SOM training was completed nearly twice as quickly as SVM-RBF. Thus, while SVM-RBF yields a higher test accuracy compared to 2SOM, the latter tends to be computationally inexpensive. To get a better sense of the performance of these classifiers, the metrics reported in Figure 5 and Figure 6 were rolled into a single performance factor which was computed as the ratio of the test accuracy to the training time. Thus, a higher test accuracy and lower training time would yield a higher performance factor. The performance factor for both classifiers is shown in Figure 6. For training data sets with up to ~20,000 observations the SVM-RBF yields a better performance factor than 2SOM. However, as data sets get larger, despite yielding lower

test accuracies, 2SOM yields better performance factor values due to significantly shorter training times.

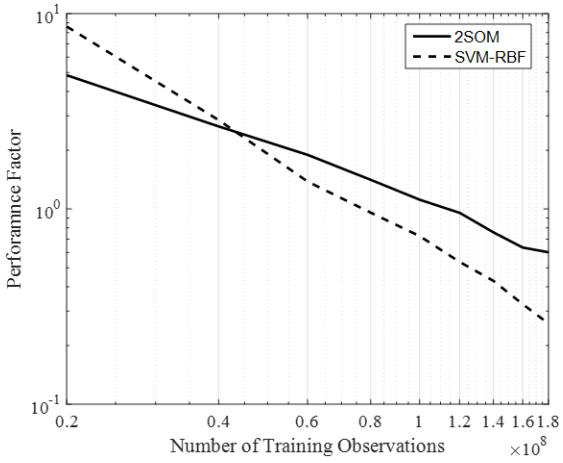

Figure 6: Performance factor computed for 2SOM and SVM-RBF as a function of increasing size of training data set (M-2SPIRAL).

This result is expected because it is known that SVMs do not scale well with increasing size of training data sets [52]. While SVMs solve the dual optimization problem to find fixed support vectors that never change after training, 2SOM seeks the equivalent of support vectors for each test point in a dynamic fashion by merely computing the nearest neighbor in each class. This idea is illustrated in Figure 7. In the case of binary classification, for every test observation, two support vectors are dynamically chosen which provide the maximum-margin for classification of that test point, thus allowing highly nonlinear decision boundaries to be constructed. The specific support vectors that provide maximum-margin must be computed for each test observation and may vary from one test observation to the next.

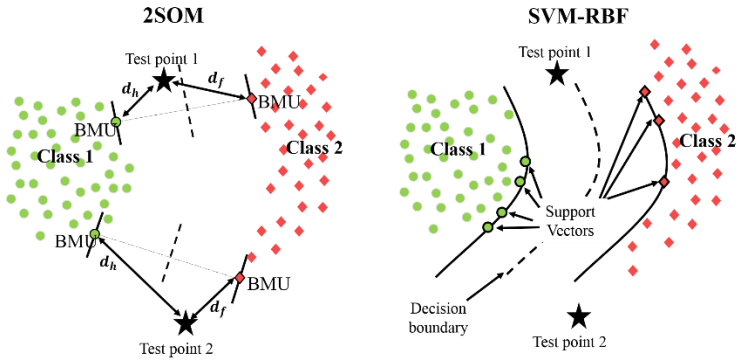

Figure 7: In the case of 2SOM, the BMUs for each test observation behave like support vectors in an SVM. In the case of SVM-RBF, fixed support vectors are discovered by solving the dual optimization problem.

However, since 2SOM requires that the distance of the test point from every prototype be computed, its algorithmic complexity varies linearly with the number of neurons used to train the SOMs. The algorithmic complexity of SVM increases quadratically as a function of the number of training observations [53]. Thus, the choice between 2SOM and SVM-RBF is a trade-off between testing accuracy and computational efficiency.

4.5. Multiclass Classification

To test the performance of cSOM on actual multiclass data sets, the almost linearly separable Iris data set was chosen. As noted in Table 1, the Iris data set consists of 4 features and 150 observations per class. This data set contains three classes, each representing a species of the iris flower. Figure 8 shows a scatter plot for this data set. In the previous section on 2SOM, two classes were randomly chosen to evaluate performance, whereas in this experimental study all three classes of data were simultaneously used for classification.

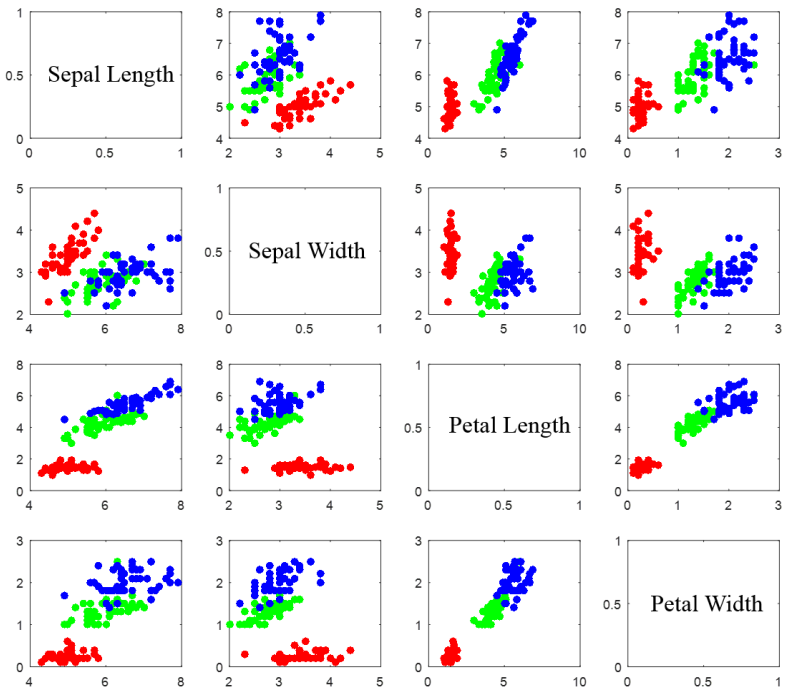

Figure 8: Scatter plot for the Iris data. Red = Setosa (Class 1), Green = Versicolor (Class 2), Blue = Virginica (Class 3).

1000 observations and 250 observations were sampled from each class for training and testing, respectively. After training one SOM for each of the Setosa, Versicolor, and Virginica classes, a test performance of ~99% was obtained. The highlight of this result was not the test accuracy, which tends to be high with even a simple linear classifier owing to the inherent separability between the classes in the data. Rather, the takeaway from this demonstration was the versatility of the developed method as a multiclass classifier. Further experiments with nonlinear, multiclass data sets are required to evaluate performance in more severe cases.

Chapter 5: The cSOM Methodology for Dealing with Imbalanced Data Sets

The imbalanced learning problem pertains to the performance of classification algorithms in the presence of underrepresented data. Most classification algorithms assume that the training data, provided as input, is balanced with regards to the classes comprising the data. When this requirement is not met, such classifiers yield unfavorable results that may be severely biased. In the context of fault detection applications, healthy data is more abundant than faulty data, resulting in naturally imbalanced data sets. Machine learning models trained on such data sets, without taking into account the class imbalance, are almost always biased towards the majority class, i.e., the healthy data. This inevitably results in a large number of type II errors, also known as false negative errors, where fault observations are incorrectly classified as healthy [54]. In the context of popular machine learning algorithms used for fault detection, SVMs are inherently sensitive to imbalance in data sets. Although solutions have been proposed as data preprocessing methods which can alleviate the class imbalance in the data before training, in their raw form, SVMs have extensively been shown to produce sub-optimal results when presented with imbalanced training data [55] [56] [57].

The cSOM methodology, provides an opportunity to alleviate class imbalance within its frameworks without having to preprocess the data. To recap, observations from each class of training data provided to cSOM are represented as neurons of a separate SOM

before classification is performed. However, before a SOM is trained, its size (the number of neurons) must be determined, and there is considerable flexibility in this regard, and this number can be arrived at experimentally. Since the trained SOM neurons substitute the training data during the actual process of classification, it is the ratio of the number of neurons in each SOM that determines the extent of imbalance prior to classification, and not the ratio of the number of original training observations. Choosing the SOM sizes prudently can alleviate the problems of a data set that was originally imbalanced. In other words, starting with an imbalanced training data set, if the SOMs are trained on the imbalanced classes using the same number of neurons, what is obtained eventually is a balanced data set, consisting of multiple SOMs (each representing a class of data) with the same number of neurons, i.e., a balanced data set for classification. This concept is first illustrated in Figure 9. A two-class problem is considered, with 1000 and 10 observations being sampled from each class of data. Then, a SOM, consisting of 100 neurons each, is trained on the two classes of data separately. A balanced data set of SOM neurons is achieved, whose patterns closely match that of the original training data set.

Performing classification, as formulated in Chapter IV, now yields better classification results. This methodology is similar to that of the undersampling and oversampling techniques used to alleviate class imbalance. In the example above, while the majority class (with 100 samples) is undersampled (to 100 neurons), the minority class (with 10 samples) is oversampled (to 100 neurons).

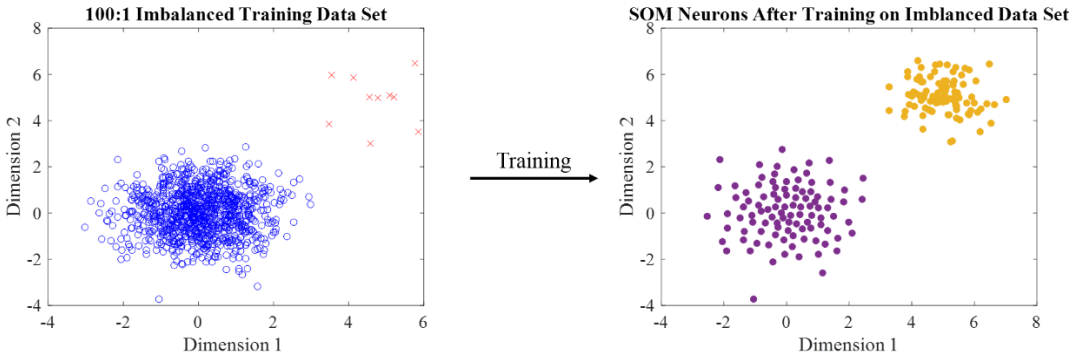

Figure 9: Illustration of how 2SOM can alleviate the imbalance problem. Training a SOM on each imbalanced class using the same number of neurons yields a balanced data set of neurons representing each class of data.

Depending on the data sets, the extent of imbalance, and the choice of the map sizes, the classes of data may be represented by more or fewer neurons than the number of observations of those classes in the original training data.

5.1. Hyperparameter Considerations for SOM

In this section, the importance of choosing the right hyperparameters for training SOMs is demonstrated with the help of an experimental study. Following from the comparison made throughout this study, the performance of 2SOM and SVM-RBF was compared on artificially generated imbalanced data sets. The 2SPIRAL data set was once again used to generate the two-class data sets. The two models were studied at different levels of imbalance ratio, starting at 1:1 (the balanced case), up to 100,000:1. The size of the majority class was fixed at 100,0000 observations. Based on the imbalance ratio, the size of the minority class was varied from 1000,000 osbervations (1:1), down to 1 observation (100:1). For each level of imbalance between 1:1 and 100:1, the 2SOM and SVM-RBF were used for classification, and the test data set was fixed at 500

observations (250 from each class) to obtain unbiased classification accuracies. Ten simulations were run at each level of imbalance to understand the deviations in the classification accuracy. In the case of 2SOM, 100 neurons were arbitrarily chosen to train the SOMs on each class of data in each case, resulting in a balanced data set for classification in every case.

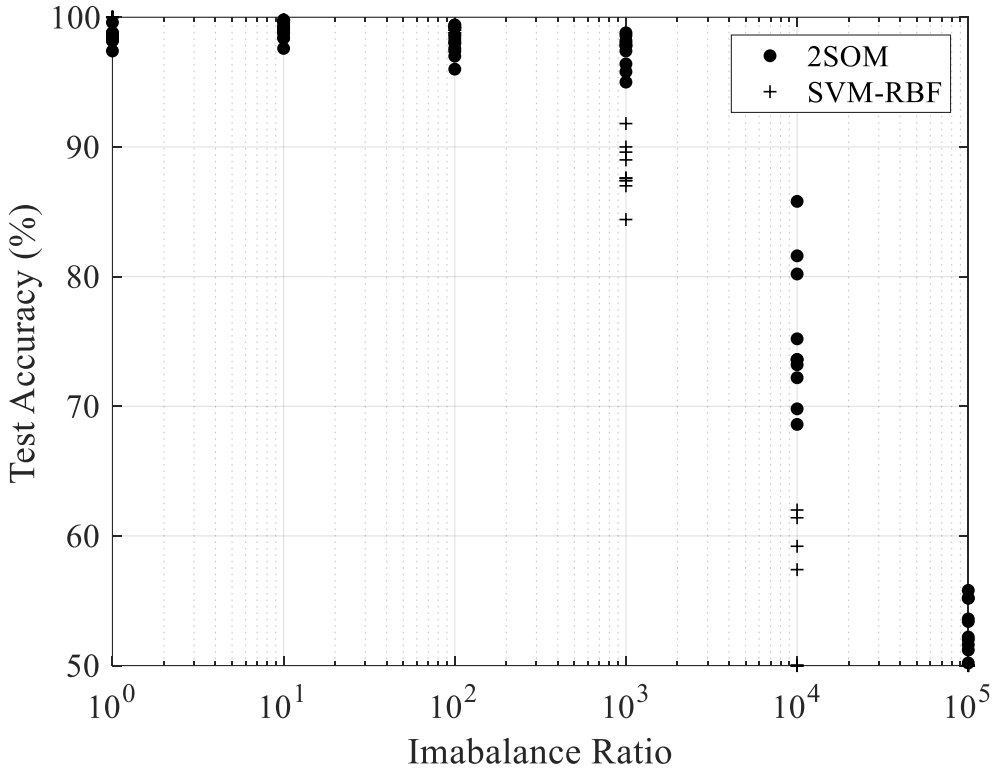

Figure 10: Test accuracy of 2SOM and SVM-RBF at different levels of imbalance of the training data set.

The results of this study are shown in Figure 10. Test accuracies for the 2SOM and SVM-RBF are shown at different levels of imbalance. Multiple experiments conducted at each imbalance ratio aid in judging the variations in the results. At low imbalance levels (1:1), the results from 2SOM and SVM-RBF are almost exactly overlapping.

This result is expected since a 1:1 imbalance ratio denotes a balanced data set. However, at the next imbalance ratio considered (6:1), the results of SVM-RBF decline, while those of 2SOM do not. Beyond an imbalance of ratio of 100:1, the results of even 2SOM start showing a decline. However, across the entire range of imbalance ratios considered for this study, the 2SOM consistently yields better test accuracies until 100,000:1, where only 1 training observation is present in the minority class. Almost none of the 10 experiments of SVM-RBF at each imbalance level yield test accuracies higher than those of the 2SOM.

It is interesting that 2SOM, despite alleviating the imbalance problem, does show a significant drop in test accuracy at the highest imbalance ratios. This drop can be attributed to the fact that at higher imbalance ratios, fewer training observations are available for training the SOM on the minority class of training data. This leads to inadequate training of the SOM and yields larger number of stray, untrained neurons at these high imbalance ratios which, in turn, lead to misclassifications. Thus, it is worth noting that the decline in performance of the two algorithms can be attributed to two completely different factors—the drop in SVM-RBF due to its inability to deal with imbalanced data sets, and the drop in 2SOM due to inadequate SOM training.

This problem can be alleviated by adjusting SOM hyperparameters such as the SOM map size, neighborhood function, learning rate, and number of training epochs. The values of these hyperparameters are important considerations to be made for a SOM to be trained well. The SOM map size, for instance, determines how many neurons must

be trained. An underestimated map size prevents the patterns in the training data to be adequately captured by the neurons. An overestimated map size on the other hand leads to several neurons to never get selected as BMUs during training, and consequently their positions remain relatively unchanged. Such a situation leads to neurons representing additional patterns, resembling noise, which may not actually exist within the training data. An underestimated value of the learning rate on the other hand leads to slow updates to the neuron positions, and does not give the neurons sufficient time to represent the training data, within the specified number of epochs.

For the experiment above, an arbitrary map size was chosen, and the default values of the other hyperparameters were used. However, a more prudent choice of these hyperparameters can improve the performance of the SOM-based methodology.

5.2. Choosing SOM Hyperparameters

This subsection discusses how the SOM hyperparameters must be chosen, in the context of the cSOM methodology, to alleviate the imbalance problem. This methodology essentially involves training several SOMs of different sizes, and using metrics that measure the SOM quality and computational efficiency to compare the various combinations of hyperparameters, and then choosing that combination of hyperparameters which yields the best combination of those metrics. The selection of the optimal map size is performed using multiobjective optimization techniques.

Discussed in this subsection are various metrics which determine the quality of the trained SOM; a multiobjective optimization strategy to empirically determine the best combination of SOM hyperparameters; and the dependence of those metrics on the SOM hyperparameters.

5.2.1. SOM Quality Metrics

As mentioned in the previous subsection, hyperparameters play an important role in how well a SOM is trained. However, there is no theoretical rule which can be used to predetermine the values of these hyperparameters. In the absence of a well-established rule based on theoretical foundations which can be used to determine an optimal map size of SOM *apriori*, the decision must be arrived at empirically [58]. Within the domain of empirical decisions, one can potentially use several different methods of achieving the same goal. For example, in the simplest case, one could train SOMs of several different map sizes and pick one size that represents the training data well even just visually. However, such a choice would be subjective. To arrive at a more robust, and quantitative decision, metrics are needed. The metrics used must be reflective of the quality of the trained SOM. In this study, two metrics based on which an optimal map size can be chosen, are considered. These metrics are – the quantization error of the trained SOM; and the difference in the information entropy of the trained SOM and the training data. These metrics are now described.

The first metric is the quantization error (QE) which is obtained by computing the average distance of the training observations to their respective BMU. The larger the value of QE, the further observations are from their BMUs, on average. The quantization error is computed as

$$QE = \frac{1}{N}\sum_{i=1}^{N}\|x_i - BMU_i\| \qquad (23)$$

where x_i is the i^{th} training observation ($i = 1, \ldots, N$), and BMU_i is the BMU corresponding to the training observation x_i. Thus, a lower QE is desirable since it suggests that the BMUs represent the training data more closely. For any data set, QE can be reduced by simply increasing the map size, thereby distributing the training samples more sparsely over the map. However, increasing the map size arbitrarily not only leads to higher training times, but may also lead to an increase in the number of stray neurons as seen in previous sections.

The second metric is the difference in information entropy of trained SOM and the training data. Information entropy (H) is defined as the average amount of information produced by a stochastic variable of data. The measure of information entropy associated with each possible value of data that the variable can take is computed as

$$H \equiv -\sum_{i=1}^{N} p(x_i)\log_2 p(x_i) \qquad (24)$$

where x_i is the i^{th} training observation ($i = 1, \dots, N$), and $p(x)$ is the probability distribution over the random variable x. Entropy is used to obtain bounds on the performance of the strongest possible lossless compression possible. This idea is very relevant in case of SOM since the representation of the training data set using a fewer number of neurons than the number of training observations, can be considered a type of data compression. Thus, comparing the information entropy of the training data set with that of the trained SOM can provide a useful metric of the information contained in the SOM. An entropy value of SOM that is closest to that of the training data set is desired. Thus, the goal must be to minimize the difference between the entropies of the trained SOM and the training data on which the SOM has been trained. Based on this the second metric is defined as

$$dH = H_{SOM} - H_{Training\ Data} \qquad (25)$$

where H_{SOM} is the information entropy of a trained SOM, and $H_{Training\ Data}$ is the information entropy of the training data set on which the SOM has been trained, and the metric dH is the difference between H_{SOM} and $H_{Training\ Data}$. Choosing hyperparameters such that the two metrics in (24) and (25) are minimized, can ensure that the SOM represents the training data well. However, both QE and dH are minimized as the map size approaches the size of the training data set i.e. a higher number of neurons is desired to arrive at a set of neurons which best represent the training data.

There may arise cases where a small loss in the above metrics may be acceptable. For instance, while performing fault detection using the cSOM methodology, a small loss in the metrics may be acceptable if the resulting SOMs yield good prediction accuracies. It is not necessary to have the best-trained SOM, if the end goal of good fault prediction accuracy can be met with a compromise in these metrics. Thus, two metrics which account for practical considerations which must be made while training SOMs for fault detection using cSOM, are considered. These metrics are the training time, and the classification accuracy on a test set. In this study it is proposed that while choosing the best combination of hyperparameters for SOM to perform fault detection using the cSOM methodology, not only the metrics in (24) and (25), but also the training time and the classification accuracy on a test set be minimized.

The classification accuracy on previously unseen test observations is a good indicator of the generalization power of a classifier. It is for this reason that this metric is considered. The aim is to choose that combination of parameters which can maximize cSOM's classification accuracy on a test set, since that is the ultimate goal of a fault detection algorithm. Minimizing the training time on the other hand can help ensure that if an acceptable classification accuracy, QE, and dH, can be achieved while saving computational resources, then that combination of hyperparameters be chosen. For instance, minimization of training time ensures that if a small map size may yield classification accuracies comparable to those of a large map, while yielding higher QE

and dH, then the hyperparameter values for the smaller map are chosen. The training time was computed as the total time (in seconds) required to train all c SOMs of cSOM.

5.2.2. Multiobjective Optimization Strategy for Choosing SOM Hyperparameters

In the absence of any theoretical or empirical rule to determine SOM hyperparameters, an optimization approach was deemed appropriate to determine the SOM hyperparameters. The SOM hyperparameters were used as the independent variables of the optimization problem who optimal values needed to be determined. The metrics described in the previous subsection were considered as potential objective functions, whose values were required to be minimized. A list of the independent variables and objective functions considered is shown in Table 4.

Table 4: List of independent variables and objective functions considered for solving the optimization problem.

Independent Variables (SOM Hyperparameters)	**Objective Functions (SOM Quality Metrics and Practical Considerations)**
SOM Map Size (N)	Difference in Entropies (dH)
Size of the Neighborhood Function (NF)	Average Quantization Error (QE)
Learning Rate (LR)	cSOM Training Time (tSOM)
-	cSOM Test Error (errSOM)

The optimization process is illustrated in Figure 11. Starting with a random combination of the hyperparameters (within the specified upper and lower bounds), the cSOM is trained. The same map size is used to train SOMs on each class of data in the training data set. This yields a balanced number of class-specific SOM neurons for further classification. The training time required to train cSOM is computed during training. After training, the entropies of the SOM neurons are compared with that of

the training data on which they were trained to obtain the value of the objective function dH. Similarly, the average quantization error QE is computed for each SOM trained. It must be noted that dH and QE are computed individually for each SOM trained.

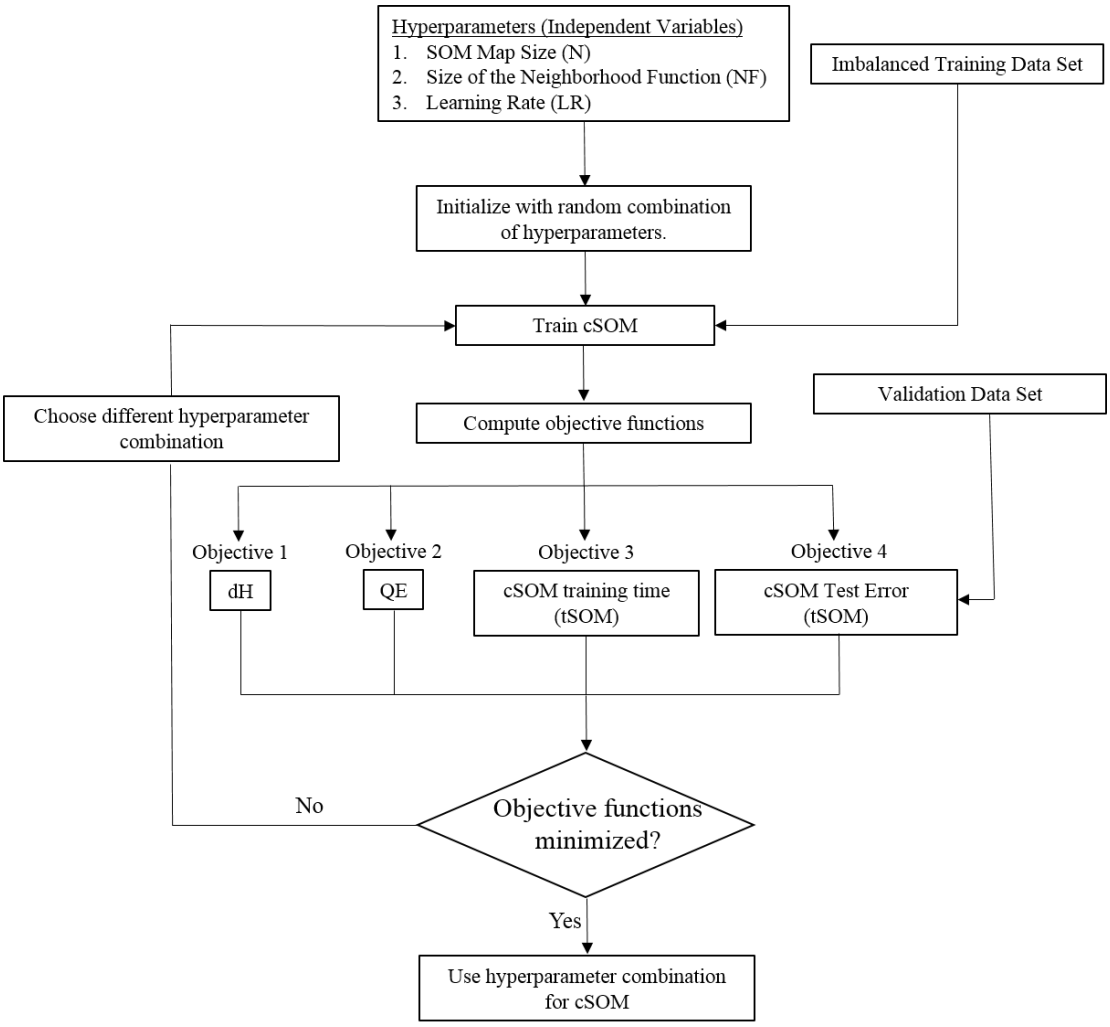

Figure 11: Multiobjective optimization strategy to determine SOM hyperparameters.

For instance, for two class problem, four values corresponding to dH and QE are computed. dH_1 and QE_1 corresponding to the SOM trained on one class of data, and

dH_2 and QE_2 corresponding to the SOM trained on second class of data. The test data set is then used to compute the classification error of the particular iteration of cSOM. Different combinations of hyperparameters are chosen, and the process repeated until minimum values of all objective functions are obtained.

The multiobjective optimization was performed using the Genetic Algorithm (GA) [59]. Genetic algorithms, are search algorithms which are based on the mechanics of natural selection. GAs determine the best individuals in a group and combine their genetic make-up to produce a new generation of individuals. In the context of selection of hyperparameters for cSOM, the individuals of a population are potential cSOM models, and the genes of each individual comprise the SOM hyperparameters used to train that individual cSOM model. The values of the objective functions (dH, QE, tSOM, and errSOM) are used to determine the best performing individuals of any generation. Then, the best individuals of a generation "reproduce' to create new individuals (cSOM models) with superior genes (hyperparameters). Random "mutation" of genes changes hyperparameters of some individuals randomly, without the influence of the "parent" individuals, allowing the optimization algorithm to stochastically search regions of the hyperparameter space, which may not have necessarily been explored by only reproduction.

5.2.3. Statistical Significance of Metrics

Before performing the experiments, it needed to be determined if the hypothesis that the chosen objective functions (dH, QE, tSOM, errSOM) affect the selection of hyperparameters, was indeed true and significant. The 2SPIRAL data set introduced in Chapter IV was to be used for the imbalanced data analysis. Thus, a Design of Experiment (DOE) was performed on the hyperparameters and the objective functions using the 2SPIRAL data set. The methodology involved training the cSOM using different combinations of hyperparameters and computing the objective functions for each case. The values of the hyperparameters used are shown in Table 5. A full factorial design was used for the hyperparameters to train different cSOMs. The full factorial design of the 7 map sizes, 5 neighborhood function sizes, and 3 learning rates, comprised 105 combinations, in total. The full-factorial DOE is presented in Appendix A. For each combination of these hyperparameters, 10 cSOMs were trained and tested to obtain the values of the six objective functions - dH_1 and QE_1 corresponding to the SOM trained on one class of data; dH_2 and QE_2 corresponding to the SOM trained on second class of data; the training time tSOM; and the classification error errSOM. In total, 1050 simulations were performed, to obtain a data set large enough to account for the variabilities in SOM training. For each cSOM, 1000 observations and 250 observations were sampled from both classes of data for training and testing, respectively.

Table 5: Values of hyperparameters used for statistical analysis.

SOM Map Size (N)	Size of Neighborhood Function (NF)	Learning Rate (LR)
5 x 5 (N=25)	1	0.01
10 x 10 (N=100)	2	0.05
15 x 15 (N=225)	3	0.1
20 x 20 (N=400)	4	-
25 x 25 (N=625)	5	-
30 x 30 (N=900)	-	-
35 x 35 (N=1225)	-	-

Having collected the data, a response surface model (second order polynomial with interactions) was fit to the data to determine the relationship between the predictors (hyperparameters), and the responses (the objective functions). The summary of fit for each response is presented in

Table 6. Further, the values of the responses as predicted by the fitted response surface model are shown in Figure 12, which provides a visual illustration of how the responses vary with the change in predictors. For instance, from Figure 12, one can see that all responses vary significantly with the predictor map size. Responses dH_1, dH_2, QE_1, QE_2, and errSOM are convex functions of the predictor map size. tSOM on the other hand monotonically increases with increase in the same predictor. The learning rate on the other hand, does not seem to have any effect on the responses, at least visually.

Table 6: Summary of fit for the computed objective functions.

Response	R-Square	R-Square Adjusted
dH1	0.9518	0.9514
dH2	0.9482	0.9477
QE1	0.6470	0.6439
QE2	0.7553	0.7532
tSOM	0.9977	0.9977
errSOM	0.5204	0.5163

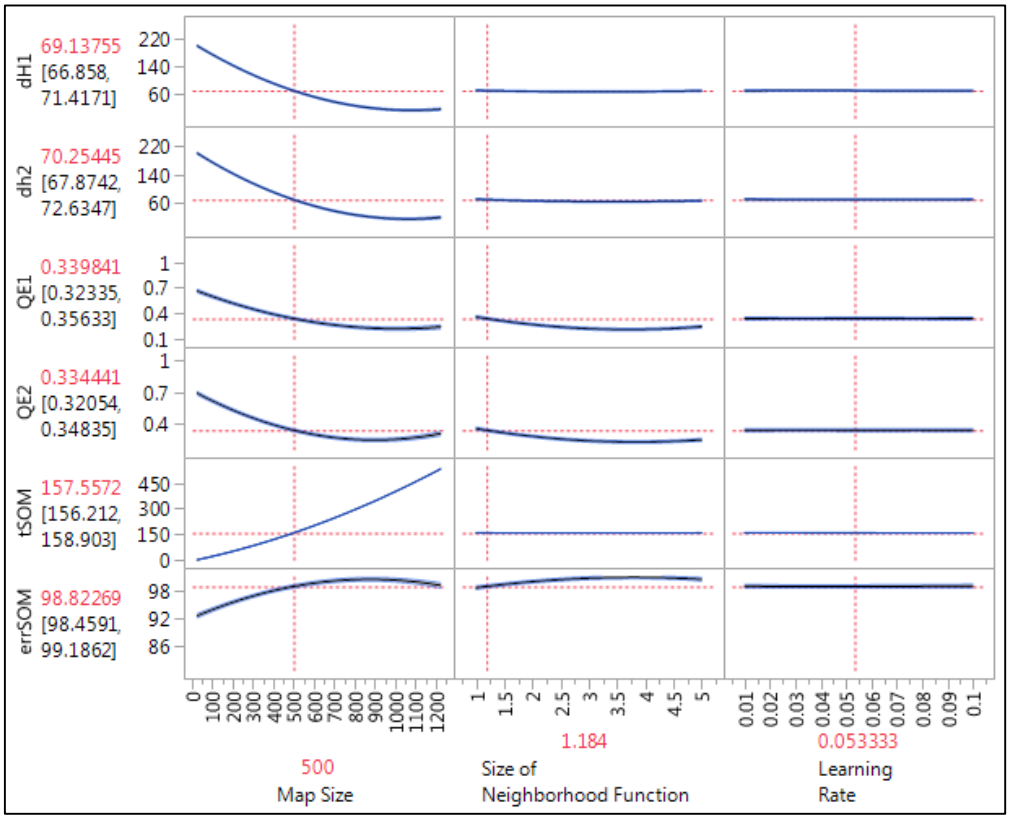

Figure 12: Profile plot of the responses as function of the predictors.

The effect of the predictors on the responses is quantified in Table 7. The source column lists different term in the response surface model, whose effects are quantified by their LogWorth. The LogWorth is the scaled version of a term's p value. It is computed as

$$LogWorth = -\log_{10}(p\ value) \qquad (26)$$

When transformed to the LogWorth scale, highly significant p-values have large LogWorths, whereas insignificant p values correspond to low LogWorths. Any

LogWorth above 2 corresponds to a p value below 0.01, and can be considered significant at the 0.01 level.

Table 7: Summary of effects of predictors on responses.

Source	p Value	LogWorth
Map Size	0.00	1264.92
Map Size*Map Size	0.00	451.327
Size of Neighborhood Function	0.00	40.116
Size of Neighborhood Function*Size of Neighborhood Function	0.00	23.465
Map Size*Size of Neighborhood Function	0.00	19.454
Size of Neighborhood Function*Learning Rate	0.48	0.317
Learning Rate	0.67	0.17
Learning Rate*Learning Rate	0.73	0.133
Map Size*Size of Neighborhood Function*Learning Rate	0.74	0.127
Map Size*Learning Rate	0.81	0.09

From Table 7, it was concluded that the map size was an important predictor, with a LogWorth of 1264.919. The same could also be concluded about the size of the neighborhood function, which had a LogWorth of 40.116. The higher-order, and interaction terms consisting of the map size and learning rate were also found to be significant. However, the same cannot be concluded about the learning rate. None of the terms in the model which had learning rate in them, including higher-order terms and interaction terms, were found to be significant, since their LogWorths were found to be well below the threshold of 2. This effect has also been visually captured in Figure 12. The values of the responses were found to not vary with the learning rate. Based on these observations, it was determined that the learning rate was not a significant hyperparameter consideration in the training of cSOM, while the map size and size of the neighborhood function were. Thus, the learning rate was not used as one of the

hyperparameters of cSOM to be optimized by the GA. The optimization would be performed on only two independent variable viz. the map size and the size of the neighborhood function.

Further, from visual inspection of Figure 12, it was found that the pairs of responses dH_1 and QE_1, and dH_2 and QE_2 showed similar trends with respect to all the predictors. Based on this observation, the relationship between these two responses was analyzed further. The correlation analysis between these two response pairs is shown in Figure 13. Figure 13 (a) depicts the response pair dh_1 and QE_1, corresponding to the SOM trained on one class of data, whereas Figure 13 (b) depicts the response pair dh_2 and QE_2, corresponding to the SOM trained on one class of data. The numbers 0.9 and 0.85 represent Spearman's rank correlation (r_s) coefficient between the corresponding responses. Spearman's rank correlation coefficient assesses how well the relationship between two variables can be described using a monotonic function. It is computed as

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \qquad (27)$$

where r_s is Spearman's rank correlation coefficient, d_i is the difference between the ranks of corresponding observations of the two variables, and n is the number of observations. The coefficient takes a high value when corresponding observations of the variables have similar ranks, and a low value otherwise. The sign of Spearman's

correlation coefficient indicates the direction of association between the two variables. A positive value indicates that one variable increases with an increase in the other, whereas a negative value indicates that one variables decreases with an increase in the other.

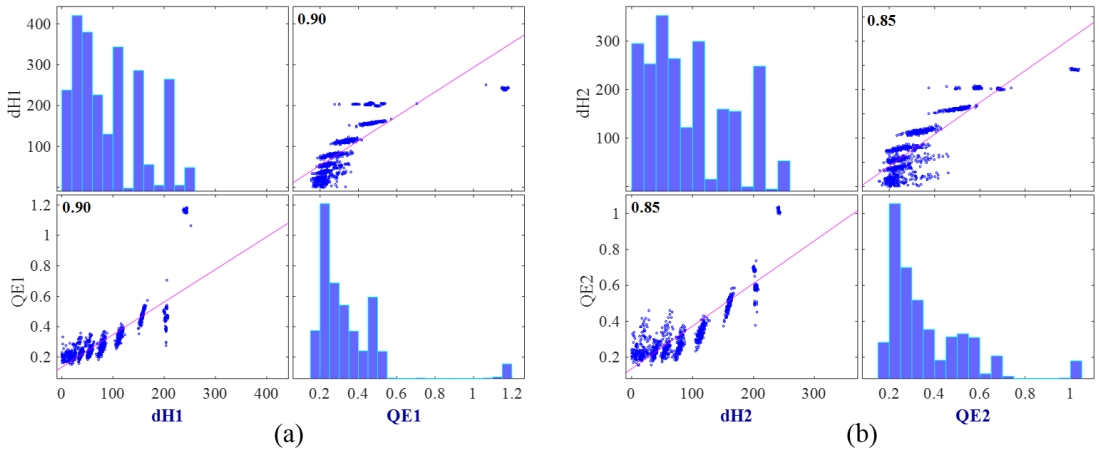

Figure 13: Correlation plots between for the two pairs of responses (a) dH1 and QE1 (b) dH2 and QE2. The numbers in the corners of the scatter plot denote the Spearman rank correlation coefficient between the corresponding responses.

$r_s = 0.9$ (for dH_1 and QE_1), and $r_s = 0.85$ (for dH_2 and QE_2) are large positive values, which indicates that the two responses dH and QE are strongly correlated. In other words, the values of dH increase with an increase in values of QE, and vice-versa. However, a high value of r_s does not imply a linear relationship between the two variables, as is evidenced by the scatter plots in Figure 13. Based on these observations, it was determined that using both dH and QE as objective functions was redundant, since the combination of hyperparameters that minimizes one would also minimize the other. Consequently, dH was eliminated from the set of objective functions to be

minimized using GA optimization. Including both would result not only in higher training times, but also in increased weights on redundant objective functions. The final list of hyperparameters and objective functions used to obtain the experimental results via GA optimization is listed in Table 8.

Table 8: List of hyperparameters and objective functions used to obtain experimental results via optimization using Genetic Algorithm (GA).

Independent Variables (SOM Hyperparameters)	**Objective Functions (SOM Quality Metrics and Practical Considerations)**
SOM Map Size (N)	Quantization Errors (QE_1 and QE_2)
Size of the Neighborhood Function (NF)	cSOM Training Time (tSOM)
-	cSOM Test Error (errSOM)

5.3. Experimental Setup and Results

In this subsection the data used to obtain the experimental results is described. How the various were designed, and how imbalance was introduced in the data is also described. Finally, the results are presented, and implications discussed. The general MATLAB code used to execute the cSOM methodology to alleviate class imbalance is presented in Appendix B.

5.3.1. Data and Design of Experiments

The 2SPIRAL data set introduced in Chapter IV was used to determine the performance of cSOM with imbalanced data. The data set consists of two classes of data, each represented by observations in one of two intertwined spirals. The data set in its original form is shown in Figure 14 (a). This data set was used as the foundation for building the different experiments. Figure 14 (b) and (c) show what the data set looks like when

complexities such as noise and imbalance are introduced to the original data set, make it more challenging to classification. For instance, noise is added to the 2SPIRAl in Figure 14 (b) which leads to more diffuse class boundaries, and increases overlap between classes as more noise is introduced. The addition of this noise was characterized by the following metric

$$d_s = \frac{\sigma}{d} \tag{28}$$

where d is the distance between the two arms of the spiral representing the two classes, and σ is the average width of the arm of each spiral. d_s represents the increase in complexity of the 2SPIRAL with the addition of noise i.e., d_s is positively correlated with the increase in noise in the data set. An increase in d_s is characterized by an increase in the thickness of the arms of the 2SPIRAL, and a decrease in the distance between the arms of the spirals representing different classes. Figure 14 (c) represents the imbalanced case. The blue observations represent the 1000 observations of the majority class, and the red observations represent the 10 observations of the minority class. The imbalanced number of observations from the minority class fail to capture the original distribution of that class of data posing challenge to correct classification.

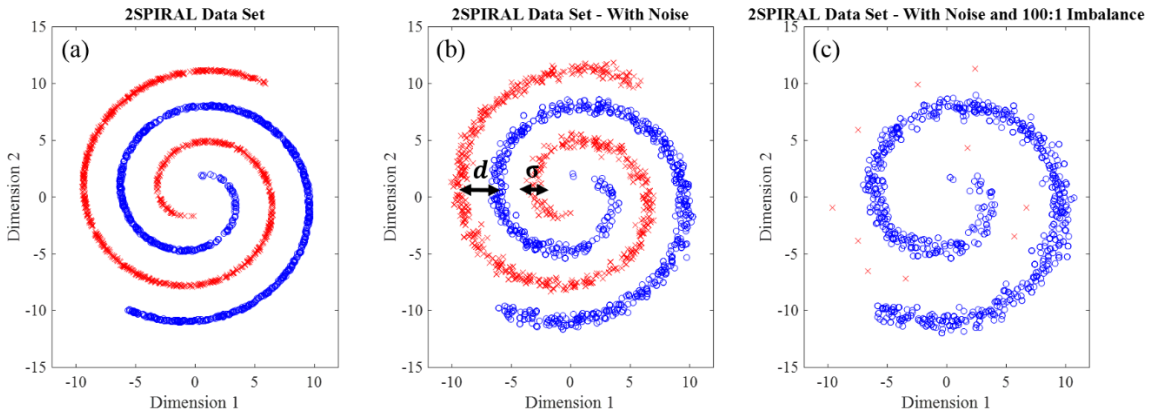

Figure 14:Various versions of the 2SPIRAL data set (a) Original data set (b) With noise introduced (c) With noise and imbalance introduced.

Based on the addition of complexities above, a set of experiments was designed to perform balancing using cSOM on 3 different levels of imbalance ratio (IR) and 3 different values of d_s were chosen to set up the experiment. These values are shown in Table 9.

Table 9: Values of Imbalance Ratio (IR) and ds chosen to setup experiments.

Imbalance Ratio (IR)	d_s
1:1	0.2
10:1	0.1
100:1	0.05

Each combination of these two parameters was considered, resulting in a total of nine experiments. In case of IR, 100:1 is the worst-case scenario, whereas in case of d_s a value of 0.2 indicates the worst-case scenario. The training data sets resulting from these nine combinations are shown in Figure 15. The worst-case scenario is represented by the case where $d_s = 0.2$ and $IR = 100:1$, due the amount of overlap between the classes, and the sparsity of data from the minority class. To prepare these data sets, the

size of the majority class was fixed at 1000 observations. Based on the imbalance ratio, the size of the minority class was varied from 1000 samples (1:1) to 10 samples (100:1). The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies.

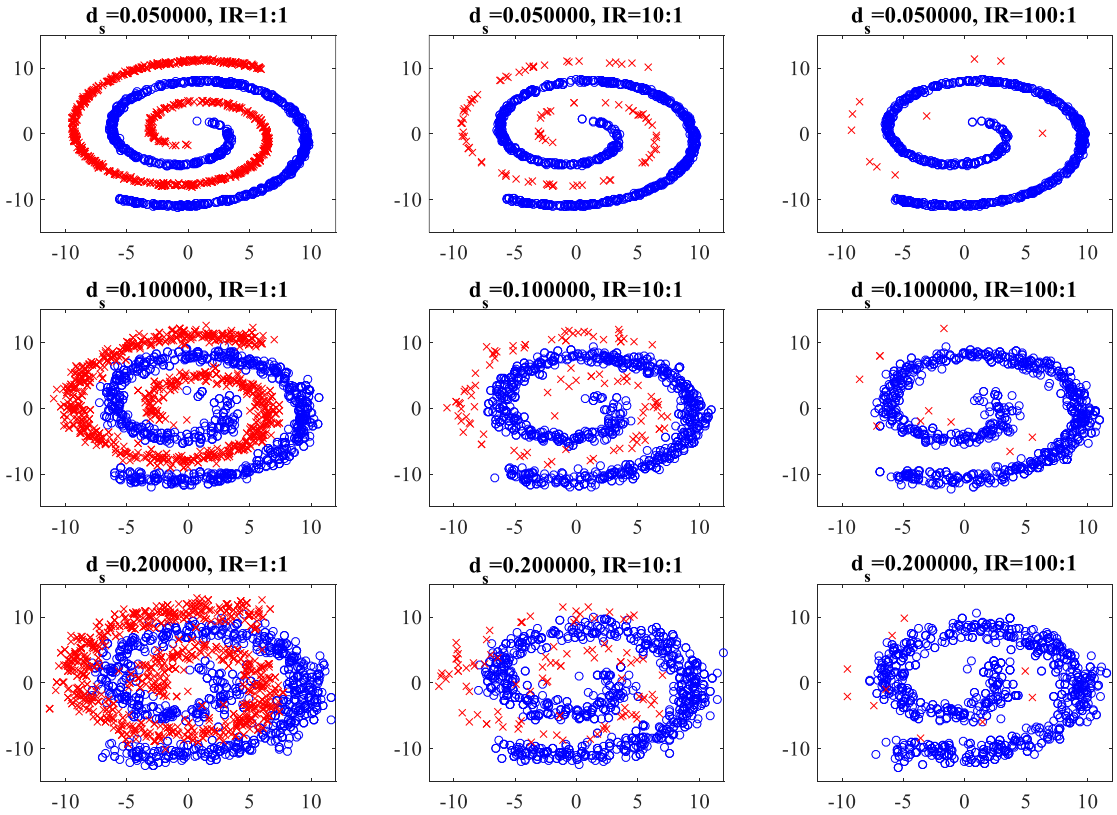

Figure 15: Data sets generated for experiments by adding noise and imbalance to the 2SPIRAL data set.

Table 10: Parameter setting for multiobjective optimization using the Genetic Algorithm (GA).

Parameter	Description	Value
Population Size	Number of individuals in a generation	**50**
Selection Function	The selection function chooses parents for the next generation based on their scaled values from the fitness functions.	**'Tournament'** (Selects each parent by choosing 2 individuals at random, and choosing the best individual out of that set to be a parent)
Crossover Fraction	The fraction individuals of the next generation that crossover produces.	**0.8**
Mutation Fraction	The fraction individuals of the next generation that mutation produces.	$= 1 - Crossover\ Fraction$ = **0.2**
Number of Generations	Stopping criteria	200

5.3.2. Genetic Algorithm Setup

For each case shown in Figure 15 i.e., for every combination of parameters in Table 9, multiobjective optimization was performed using the Genetic Algorithm (GA). The GA, based on the four objective functions, was used to provide the best combination of the SOM hyperparameters to use to train cSOM to achieve a high test accuracy, while choosing the smallest map size to balance the two classes of data. This section provides descriptions of the parameters used to tune GA, along with their values chosen for to obtain the experimental results. These parameters, their description, and the value chosen are listed in Table 10.

5.3.3.Results

To recap, the cSOM was used on each of the cases shown in Figure 15, to balance the data and then perform classification. The 3 levels of d_s and IR constituted a total of 9

different data sets, with varying levels of noise and imbalance. The size of the majority class was fixed at 1000 observations. Based on the imbalance ratio, the size of the minority class was varied from 1000 samples (IR = 1:1) to 10 samples (IR = 100:1). The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies. The Genetic Algorithm optimization was used to determine the optimal number of neurons to be used to obtain a balanced data set for classification. A summary of these results in provided in Table 11.

In each of the 9 cases, regardless of the level of noise or imbalance, the optimization algorithm opted for the largest or the second largest map size possible. This can be attributed to the fact that the optimization algorithm prioritizes the three objective functions (QE_1, QE_2, and errSOM) which require large map sizes to be minimized, over the other objective function (tSOM) which requires as small a map size as possible to be minimized.

Further, the increase in classification error with an increase in imbalance ratio was found to be greater than the increase in classification error with an increase in d_s. It was interesting to find that for imbalance ratios 1:1 and 10:1, the classification error increased by a factor ~2 with increasing values of d_s, but in the case of imbalance ratio of 100:1 the classification error first decreased and then increased with increasing values of d_s. This may be due to the fact that at very high levels of imbalance ratios, where only 10 observations are available from the minority class, the effect of addition of noise is negligible. In other words, when the number of observations available is as

Table 11: Summary of experimental results cSOM (N – Map Size, NF – Size of Neighborhood Function, QE – Quantization Error).

	d_s=0.05, IR=1:1	d_s=0.05, IR=10:1	d_s=0.05, IR=100:1
N	289	289	225
NF	2	2	2
QE_1	0.294	0.253	0.306
QE_2	0.303	0.303	1.79
tSOM (s)	155.30	86.57	60.24
errSOM (%)	1.95	3.76	25.18
	d_s=0.1, IR=1:1	**d_s=0.1, IR=10:1**	**d_s=0.1, IR=100:1**
N	225	25	25
NF	3	2	5
QE_1	0.238	0.154	0.155
QE_2	0.211	0.396	1.600
tSOM (s)	113.73	8.08	7.25
errSOM (%)	4.55	5.31	23.11
	d_s=0.2, IR=1:1	**d_s=0.2, IR=1:1**	**d_s=0.2, IR=100:1**
N	289	196	196
NF	3	3	4
QE_1	0.185	0.207	0.213
QE_2	0.198	0.407	0.171
tSOM (s)	155.12	54.49	49.54
errSOM (%)	5.62	11.79	27.86

small as 10, the representation of the original class of data is so poor that upon adding noise to such data, the original patterns are actually better represented.

5.3.4. Repeatability of Optimization and Sensitivity Analysis

Reproducing results when using stochastic algorithms such as GA, is a known concern. In this context, the factors governing the repeatability of the results obtained using GA are discussed in this subsection.

Table 12: Lower and upper bounds on the independent variables provided as input to the GA.

Independent Variables (SOM Hyperparameters)	Lower Bound	Upper Bound
SOM Map Size (N)	5x5	35x35
Size of the Neighborhood Function (NF)	1	5

Given that the training data remains same, there are two main factors which may lead to difference in the final optimized hyperparameters, across different GA simulations. The first of these two factors is the upper and lower bounds on the hyperparameters. The GA takes as input the upper and lower bounds for each of the independent variables (the hyperparameters). The bounds used in this analysis are shown in Table 12. The lower bound on the SOM map size was chosen as 5x5 to yield a total of 25 neurons. This selection allows the minority class to be represented by more neurons than the number of training observations in the worst-case scenario (100:1 imbalance). The upper bound of 35x35 yields 1225 neurons, which allows the majority to be represented by only 225 more neurons than training observations. The rationale is that it does not make sense that the SOM representation of training data contain more prototypes than the number of training observations. Setting the upper bound on the neighborhood function as 5 ensures that even for the smallest map size (5x5), the size of the neighborhood function does not exceed the size of the SOM in either direction. The lower bound on the neighborhood function is just the minimum possible value of 1. Since these bounds define the search space for the GA, providing a different range of

bounds may lead to different ‘optimal’ values of the independent variables across different GA simulations.

The other, more important factor, which governs the reproducibility of results of GA is the GA initialization. The GA initialization determines how the individuals of the first generation are created. To recap, the individuals of a generation are the potential cSOM models generated by different hyperparameter combinations. The choice of hyperparameters in the first generation has significant influence on the final optimal combination of hyperparameters, since the subsequent generation models are majorly comprised of some combination of individuals from the previous generation. Thus, the region of the hyperparameter space in which the GA is initialized influences the final choice of hyperparameters. By default, the GA uses random initialization of the first generation of individuals i.e. the first generation consists of models with random hyperparameter combinations, chosen within the region of the hyperparameter space defined by their upper and lower bounds. Alternately, the individuals of the first generation can be specified manually, as input to the GA i.e. a set of fifty, user-defined hyperparameter combinations to serve as the starting points for the GA.

The results in the previous section were arrived at by manually specifying the first generation individuals for the GA. The fifty hyperparameter combinations used to initialize the GA manually are listed in Appendix C.

A set of analysis was also performed using the random initialization, on the artificial data sets shown in the previous subsection. All the data and settings were retained, only the initialization was changed to random, as against specifying manually. The results

obtained using these two different initialization methods are compared in Table 13. The results show that the most important objective function for fault detection, the test error, is not adversely affected by the change in initialization strategy. While in some cases, the manual initialization yields lower error rates than random initialization, in some cases it does not. In either case, the difference in the test errors from both initializations does not differ by more than a ~3-4%. This suggests that the surface of the objective function 'errSOM' with respect to the hyperparameters is relatively flat, and riddled

Table 13: Results obtained on artificial data sets using manual and random initializations of GA.

	d_s=0.05, IR=1:1		**d_s=0.05, IR=10:1**		**d_s=0.05, IR=100:1**	
	Manual	**Random**	**Manual**	**Random**	**Manual**	**Random**
N	289	1089	289	676	225	1089
NF	2	2	2	4	2	3
QE_1	0.294	0.200	0.253	0.233	0.306	0.21
QE_2	0.303	0.210	0.303	0.130	1.79	2.610
tSOM (s)	155.30	750	86.57	300	60.24	450
errSOM (%)	1.95	4.05	3.76	3.16	25.18	23.54
	d_s=0.1, IR=1:1		**d_s=0.1, IR=10:1**		**d_s=0.1, IR=100:1**	
	Manual	**Random**	**Manual**	**Random**	**Manual**	**Random**
N	225	529	25	25	25	676
NF	3	2	2	2	5	2
QE_1	0.238	0.180	0.154	0.160	0.155	0.180
QE_2	0.211	0.190	0.396	0.410	1.600	3.760
tSOM (s)	113.73	350	8.08	7.5	7.25	250
errSOM (%)	4.55	2.5	5.31	1.5	23.11	18.36
	d_s=0.2, IR=1:1		**d_s=0.2, IR=1:1**		**d_s=0.2, IR=100:1**	
	Manual	**Random**	**Manual**	**Random**	**Manual**	**Random**
N	289	400	196	400	196	400
NF	3	3	3	2	4	3
QE_1	0.185	0.160	0.207	0.160	0.213	0.170
QE_2	0.198	0.160	0.407	0.55	0.171	2.72
tSOM (s)	155.12	225.78	54.49	125.93	49.54	90.26
errSOM (%)	5.62	4.48	11.79	13.67	27.86	25.24

with local minima. The other objective functions are, however, significantly impacted by the choice of initialization. For instance, QE_2 (the quantization error of the SOM trained on the minority class of training data), at higher imbalance ratios, takes lower values with the manual initialization than the random. This can be attributed to the fact that the manual initialization, which yields smaller map sizes than random initialization, allows fewer ill-trained neurons to be present in the SOM, leading to a reduction in the average quantization error. Similarly, tSOM (the training time of SOM), is also lower for the manual initialization due to the smaller map sizes.

5.3.5. Comparison with Other Classifiers

To validate the effectiveness of cSOM, its performance on the imbalanced and noisy 2SPIRAL data set was compared with that of two other classifiers. Following from the comparison throughout this study, SVM-RBF was used as one of the classifiers. The other classifier chosen for comparison was a neural network (NN) with one hidden layer. The three classifiers were tested on the nine data sets used in the previous subsection. The nine data sets comprised different levels of noise and imbalance in the 2SPIRAL data set. As in the previous subsection, based on the imbalance ratio, the size of the minority class was varied from 1000 samples (IR = 1:1) to 10 samples (IR = 100:1). The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies.

The hyperparameters of SVM-RBF were optimized using a built-in MATLAB library which attempts to minimize the cross-validation error by varying SVM's

hyperparameters. For the neural network to achieve best performance, the size of its hidden layer was varied from 10 neurons to 100 neurons, and the classification error recorded for each. For each data set, the result corresponding to the hidden layer size yielding the lowest classification error was reported. Additionally, in each case, the reported test error for these classifiers was the mean test error obtained by training and testing each classifier ten times. For cSOM, the reported test errors were the same as the ones reported in Table 11. The comparative results from these experiments is shown in Figure 16, Figure 17, and Figure 18. In each of these plots, the x-axis represent different levels of complexity achieved by the addition of noise, as measured by d_s. Figure 16, Figure 17, Figure 18 correspond to the three levels of imbalance (1:1, 10:1, and 100:1 respectively) at which the classifiers were evaluated.

At low levels of imbalance i.e., when the data set was perfectly balanced (Figure 16), NN yielded the lowest test errors (compared to cSOM and SVM-RBF) at all three levels of d_s. For the simplest case where $d_s = 0.05$, NN achieved 0% classification error on the test data set while using 80 neurons in the hidden layer. This is understandable, since NNs are known to be powerful classifiers when the training data is balanced and nonlinearly separable, and can achieve good generalization on a previously unseen test data set. The performance of cSOM was marginally poorer than that of SVM-RBF for $d_s = 0.05\ and\ 0.1$. However, for the classification accuracy of cSOM for the case where $d_s = 0.2$ was better than cSOM by ~3%. All three classifiers yielded comparable results for this case. Even at an imbalance ratio of 10:1 (Figure 17), for low levels of

noise ($d_s = 0.05$), NN yielded 0% error on the test data set, however, this time using 70 neurons in its hidden layer, as compared to the balanced case where it used 20 neurons in the hidden layer. With the addition of the additional complexity of an imbalanced training data set, the performance of all three classifiers suffered. The trends however were found to be different from the balanced case. While NN was still the best-performing classifier amongst the three, cSOM yielded, on an average, ~17% lower classification errors than SVM-RBF, across the three levels of noise. The performance of SVM-RBF was found to suffer the most with the increase in imbalance, but NN stood out as the best classifier even though its classification error increased marginally upon increasing the imbalance ratio. At the highest level of imbalance ratio considered (IR=100:1 in Figure 18), however, the trend in performance on the three classifiers changed significantly from previous levels of imbalance considered. The performance of all three classifiers suffered at an imbalance ratio of 100:1. Across all three levels of d_S, the classification errors of NN and SVM-RBF increased by an average of ~26% and ~20% respectively, from the case of 10:1 imbalance. The classification error of cSOM, however, increased by ~10%. cSOM was also found to be the best-performing classifier across all three levels of d_s. It achieved classification errors lower than that of NN and SVM-RBF by ~12% and ~26% respectively. Also, NN yielded classification errors of 33% and 35% for $d_s = 0.1\ and\ 0.2$ respectively, despite having used 100 neurons in its hidden layer.

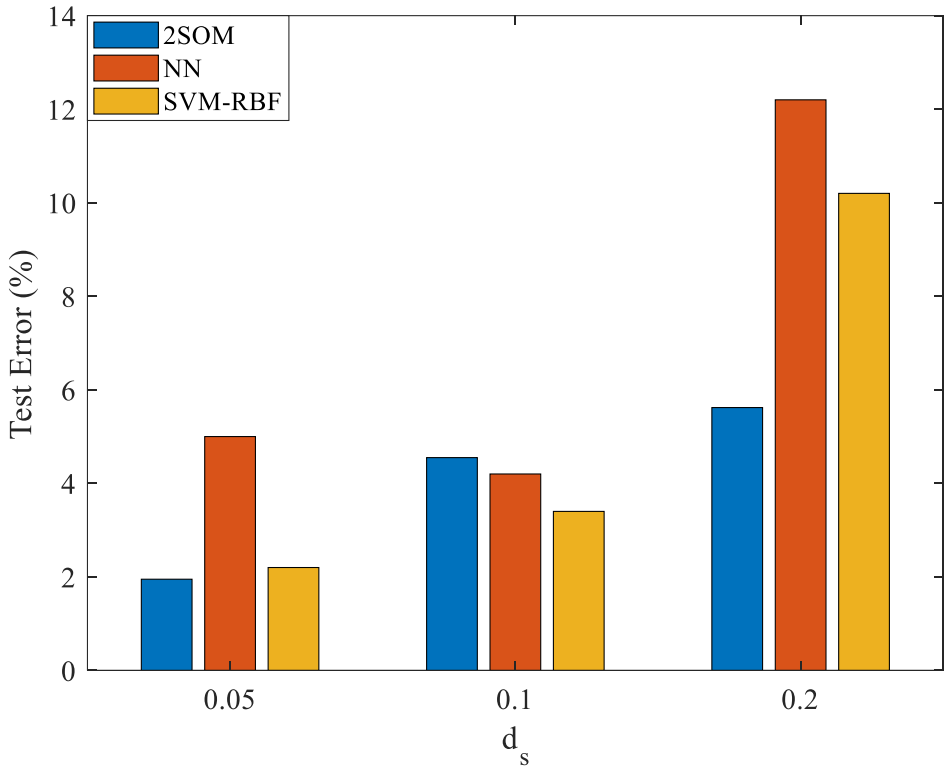

Figure 16: Test errors of cSOM, Neural Network, and SVM-RBF at different levels of noise on the 2SPIRAL data, with an imbalance ratio IR = 1:1.

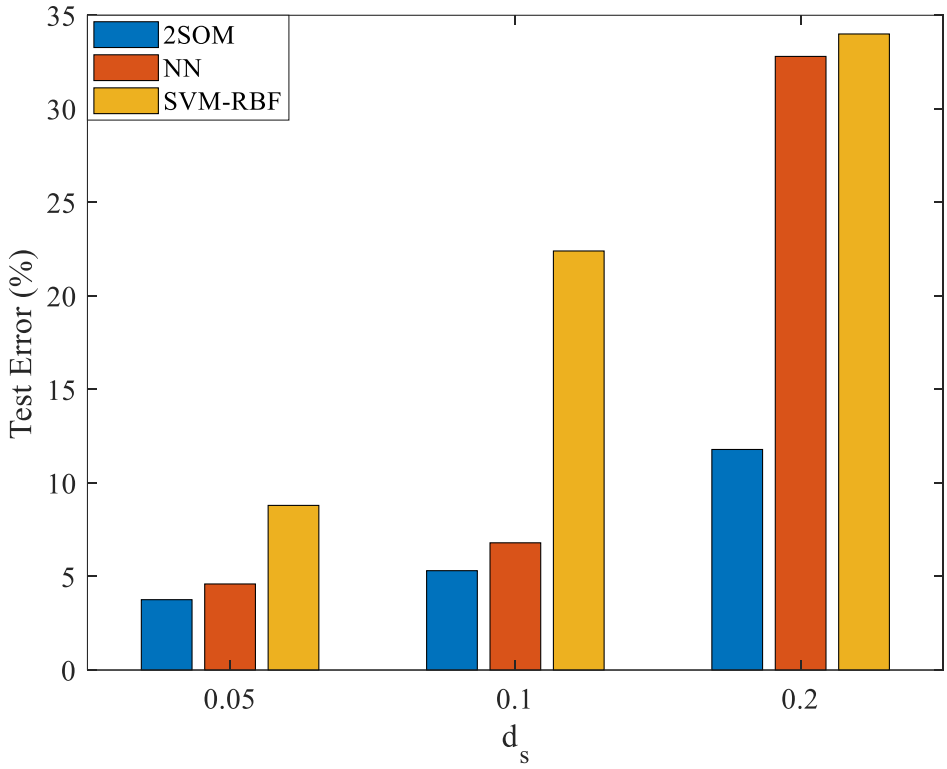

Figure 17: Test errors of cSOM, Neural Network, and SVM-RBF at different levels of noise on the 2SPIRAL data, with an imbalance ratio IR = 10:1.

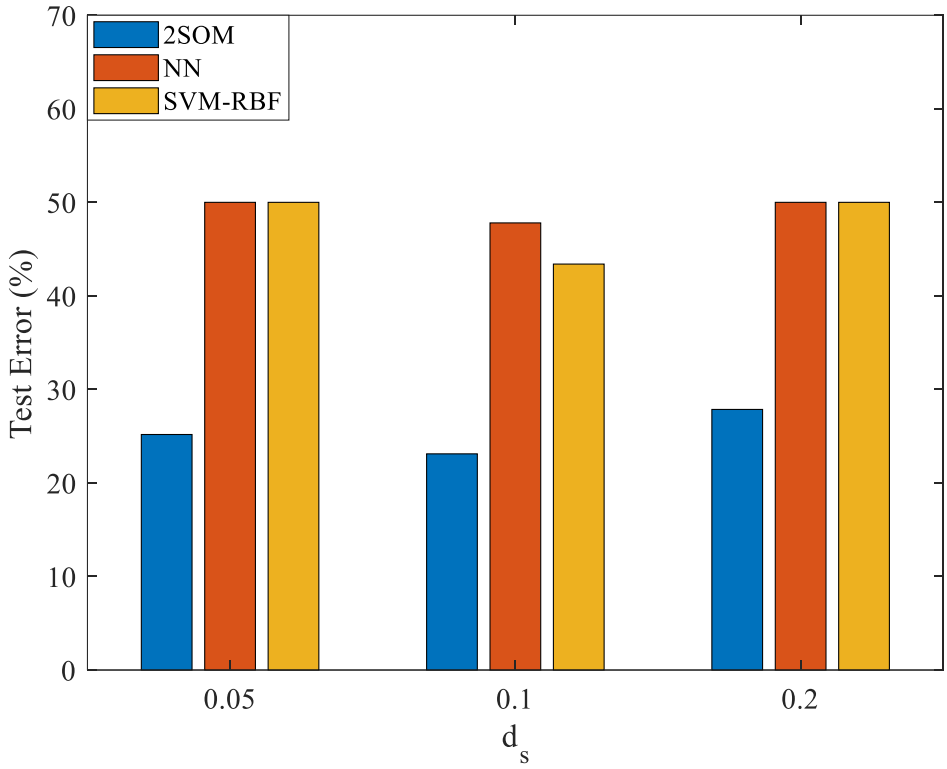

Figure 18: Test errors of cSOM, Neural Network, and SVM-RBF at different levels of noise on the 2SPIRAL data, with an imbalance ratio IR = 100:1.

The performance of all three classifiers was found to be sensitive to the addition of noise to the data set. This was expected since the addition of noise leads to an increase in the overlap between distributions of the two classes. However, the change in the classification errors of all three classifiers with a change in the imbalance ratio (across all three levels of d_s) was found to be greater than the change in their classification errors with a change in d_s (across all three levels of imbalance ratio). In other words, it was found that the imbalance ratio affected the performance of cSOM, NN and SVM-

RBF on an average by ~11%, ~20% and ~28% respectively, while the addition of noise affected their performance, on an average by ~5%, ~6% and ~8% respectively.

The fact that the performance of cSOM is altered by ~11% by the imbalance ratio (nearly half as that of NN, and third as that of SVM-RBF) is proof that the developed cSOM methodology is robust when it comes to tackling imbalanced data sets for classification.

5.3.6. Experiments Using Benchmark Data Sets

To validate the effectiveness of cSOM using benchmark data sets, its performance was compared with that of two other classifiers. Following from the comparison throughout this study, SVM-RBF was used as one of the classifiers. The other classifier chosen for comparison was a neural network (NN) with one hidden layer. The three classifiers were tested on the four benchmark data sets used in Section 4.4.1 As in the previous subsection, based on the imbalance ratio, the size of the minority class was varied from 1000 samples (IR = 1:1) to 10 samples (IR = 100:1). The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies.

The hyperparameters of SVM-RBF were optimized using a built-in MATLAB library which attempts to minimize the cross-validation error by varying SVM's hyperparameters. For the neural network to achieve best performance, the size of its hidden layer was varied from 10 neurons to 100 neurons, and the classification error recorded for each. For each data set, the result corresponding to the hidden layer size

yielding the lowest classification error was reported. Additionally, in each case, the reported test error for these classifiers was the mean test error obtained by training and testing each classifier ten times. . The comparative results from these experiments is shown in Figure 19, Figure 20, Figure 21, and Figure 22.

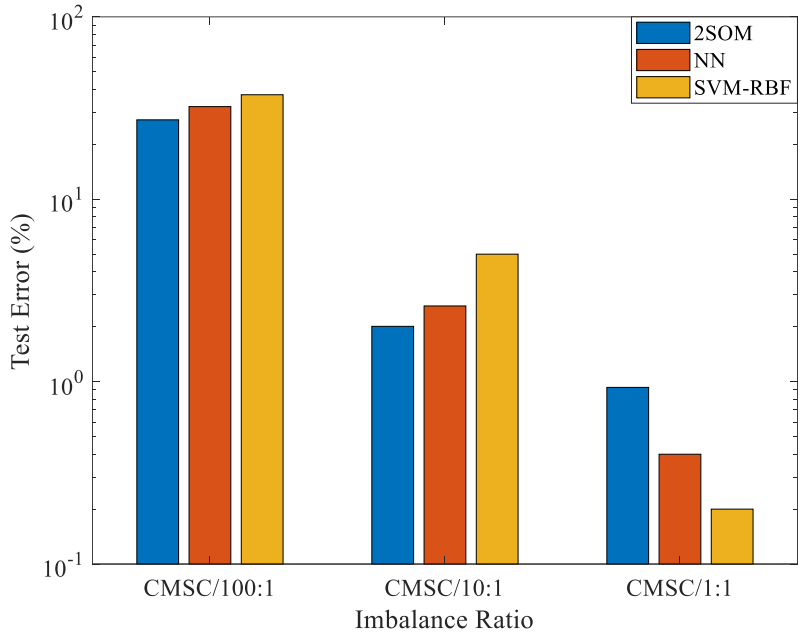

Figure 19: Test errors of cSOM, Neural Network, and SVM-RBF for the CMSC data set, at 1:1, 10:1, and 100:1 imbalance ratios.

As in the case of experiments with artificial data sets, at lower imbalance ratios, the performance of NN and SVM-RBF on the benchmark data sets were comparable to, or better than that of cSOM. However, at higher imbalance ratios, the performance of cSOM was found to be better than that of NN and SVM-RBF. This was true for the CMSC and Iris data sets, but not for the Wine and BCW data sets, where at 100:1 imbalance, the performance of neural networks was found to better, even if by only a few percentage points.

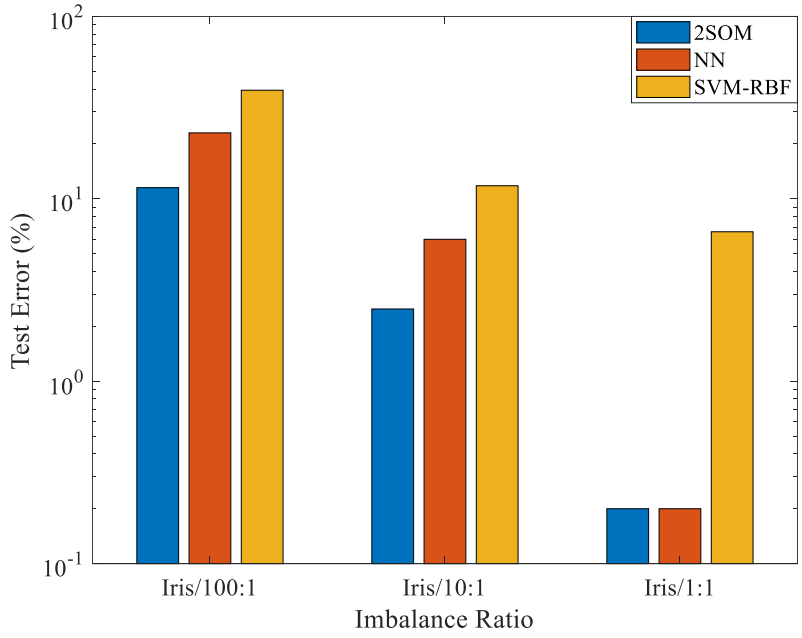

Figure 20: Test errors of cSOM, Neural Network, and SVM-RBF for the Iris data set, at 1:1, 10:1, and 100:1 imbalance ratios.

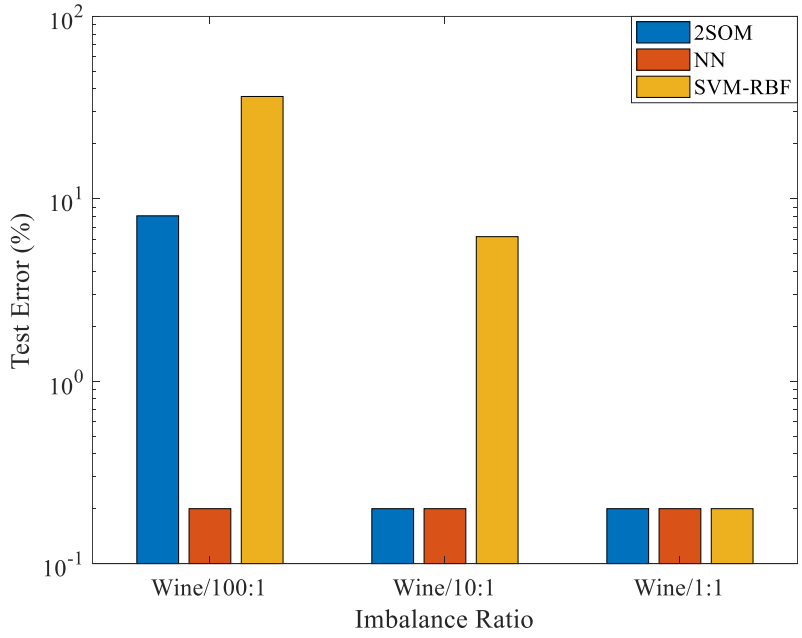

Figure 21: Test errors of cSOM, Neural Network, and SVM-RBF for the Wine data set, at 1:1, 10:1, and 100:1 imbalance ratios.

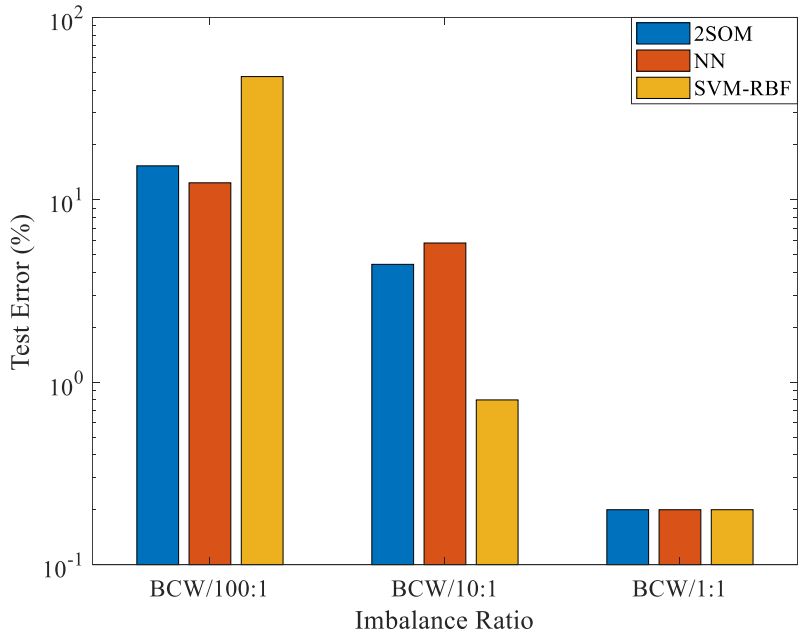

Figure 22: Test errors of cSOM, Neural Network, and SVM-RBF for the BCW data set, at 1:1, 10:1, and 100:1 imbalance ratios.

Confusion matrices depicting the performance of all three classifiers, on artificial and benchmark data sets are provided in Appendix D.

Chapter 6: The Semi-Supervised cSOM Methodology for Dealing With Sparsely-Labeled Data Sets

Supervised learning methods require large amounts of labeled data to train meaningful models. However, labeled condition monitoring data is often difficult and expensive to acquire in large quantities, due to the costs involved, and the manual input from experts required. Thus, supervised learning techniques are not useful in such scenarios.

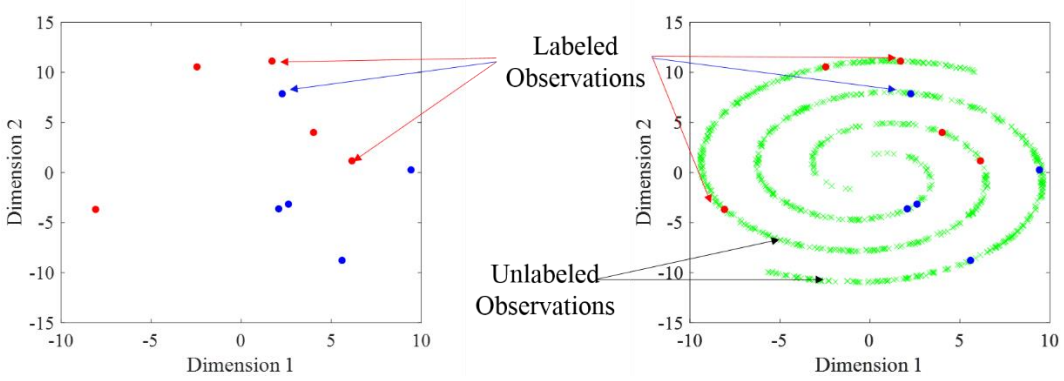

Figure 23: Issues with learning from only few labeled observations, as compared to learning from both labeled and unlabeled observations.

Unlabeled observations, on the other hand, are available in much larger quantities than labeled observations. Despite lacking class labels, large volumes of unlabeled data can contain significant information about the healthy and faulty operating states of a system, which cannot be explored by supervised learning techniques. Consider the simple situation in Figure 23. When only a few labeled observations are available, they do not provide adequate information regarding the underlying distribution of the data. Building a supervised classifier on few labeled observations lead to incorrect decision

rules being learnt. Upon looking at the unlabeled observations, however, provides a clearer picture of the class distributions, despite lacking label information. When only a small number of labeled observations but considerably larger number of unlabeled observations are available, the semi-supervised learning methods can be considered cost-effective alternatives to supervised and unsupervised learning methods.

6.1. Introduction to Semi-Supervised Learning

Semi-supervised learning techniques are machine learning algorithms which lie at the confluence of supervised and unsupervised learning techniques. These techniques use both, labeled and unlabeled data in conjunction to develop classifiers, to perform an otherwise supervised learning task. The goal of semi-supervised learning is to build a classifier which has better prediction accuracy on previously unseen test observations than a supervised learning technique would have achieved using just the limited number of labeled observations. A mapping: $X \mapsto Y$, where X is the set of training observations, and Y the set of labels corresponding to those observations, is required to be learnt from the labeled and unlabeled data. The mapping f can be learnt from unlabeled observations by making certain assumptions about the relationship between the marginal distribution of the unlabeled data $P(X)$ and the conditional distribution $P(Y|X)$ which determines the class labels Y.

There exist two different semi-supervised learning settings – inductive and transductive semi-supervised learning. The goal of both settings is slightly different.

Inductive Semi-Supervised Learning: The goal of inductive semi-supervised learning is to predict the labels of future test observations. Given a training data set containing l labeled observations $\{(x_i, y_i)\}_{i=1}^{l}$, and u unlabeled observations $\{x_j\}_{j=l+1}^{l+u}$, inductive semi-supervised learning aims to learn a function $f: X \mapsto Y$ such that f can be a good predictor for future test observations.

Transductive Semi-Supervised Learning: The goal of transductive semi-supervised learning is to predict the labels of the unlabeled observations in the training data. Given a training data set containing l labeled observations $\{(x_i, y_i)\}_{i=1}^{l}$, and u unlabeled observations $\{x_j\}_{j=l+1}^{l+u}$, transductive semi-supervised learning aims to learn a function $f: X^{l+u} \mapsto Y^{l+u}$ such that f can be good predictor for the unlabeled observations $\{x_j\}_{j=l+1}^{l+u}$.

6.2. Developed Semi-Supervised cSOM Methodology

The semi-supervised methodology developed in this study is transductive in nature. The goal is to utilize the information available from the labeled training observations to first characterize the class distributions in the training data. Then, with the class membership information available for the entire training data set, classification is performed on a set of previously unseen test observations. However, even prior to labeling, the distribution of the training data, including unlabeled data, must be characterized. Thus, the developed methodology is divided into three parts. First, the

clustering is performed on the training data set to characterize its distribution by generating prototypes of all training observations. Then, the labeled observations are used to predict the class membership of all prototypes. Finally, based on the similarity of unseen test observations and the labeled prototypes, the class membership of the test observations is inferred. Each of these steps is described in detail in the subsection that follow.

6.2.1.Clustering

As stated before, certain assumptions must be made about the relationship between the marginal distribution of the training data $P(X)$, and the conditional distribution $P(Y|X)$ of the class labels. A common assumption in the semi-supervised learning paradigm is that observations closer to each other, or those that belong to the same cluster of data, are likely to be similar and thus share the same class label [60] [61] [62]. Following from this assumption, self-organizing maps are used for the clustering step in this study. The SOM-based classifier developed in Chapter 4, and the SOM-based methodology to alleviate class imbalance problems developed in Chapter 5 used SOMs in a supervised manner i.e., in those methods SOMs were trained on specific classes in the training data. However, for the semi-supervised methodology, the SOM is used in its traditional, unsupervised form for clustering. The hyperparameters of the SOM can still be determined using the optimization strategy developed in Chapter 5. However, the training must be done in an unsupervised manner, and metrics such as the difference in entropy of training data set and SOM neurons (dH) and the average quantization error

(QE) can be used as the objective functions to be minimized. The test error, however, cannot be used as one of the objective functions here, while the training time can still be used if it is a concern.

6.2.2.Labeling

Once the SOM has been adequately trained, its neurons are prototypes of the training data on which it was trained. The neurons however, unlike in the case of the supervised cSOM, do not have labels assigned to them i.e., their class membership is unknown. To be able to perform classification on test observations, the class membership of the neurons must be known. This labeling can be usually be performed using traditional supervised classification using the labeled observations as the training data, and the SOM neurons as the test data. However, traditional classifiers (which perform 'hard' classification) result in assignment of a single class label to every neuron, without regard for their proximity to the labeled observation. There is a degree of uncertainty involved in the labeling neurons since the training observations are scarce, and in such cases class memberships cannot be predicted with absolute certainty. Thus, it is more prudent to assign a degree of class membership to each neuron, rather than single class labels i.e. soft classification. In the developed methodology, class memberships of neurons are computed based on the distance of each neuron from its nearest labeled training observation of each class. In fact, this is the same metric that is computed to perform multiclass classification using cSOM. Thus, for a c-class problem, each neuron is assigned c membership values, each representing the degree of similarity of a neuron

to each class of data. This class membership value, which is referred to as its 'weight' for each class is computed as

$$W_j^i = 1 - \frac{d_j}{\sum d_j} \qquad (29)$$

where W_j^i is the weight of the ith neuron for the jth class, d_j is its Euclidean distance from the nearest labeled training observation in class j, and denominator is the sum of its Euclidean distances from the nearest neighbor in each of the j classes. Thus, if a neuron that is closest to class j, its weight value W_j^i is the highest for class j, and lowest for the class from which it is farthest. Thus, computing class memberships this way allows each neuron to have a 'degree' of membership to each class. For neurons which lie far from labeled training observations in the Euclidean space, the weight values are nearly equal for every class and thus naturally represent an uncertainty of them belonging to any class of data.

It must be noted that the developed method requires at least one labeled training observation from each class of training data to be implemented. Without this requirement being met, weights cannot be assigned to the neurons for the class of training data which does not have any labeled training observation.

6.2.3.Classification

Once class membership weights have been assigned to the neurons, classification can be performed on previously unseen test observations. However, since the neurons are assigned class membership weights, and not class labels, traditional classifiers cannot

be used. Instead, a variant of the k-nearest neighbors algorithm is used for classification of test observations. In the classic k-nearest neighbors algorithm, the k-closest training observations to the test observation are selected, and the class to which majority of those training observations belong is also assigned to the test observation. In this study however, the class membership weights of the k-nearest neighbors of a test observation are summed, and the test observation is assigned to that class which has the maximum sum of weights for the k-nearest neighbors. Also, the k-nearest neighbors of the test observation are the neurons of the trained SOM, rather than the training observations themselves. An example of how a test observation is classified is provided in Table 14. If for a test observation $\overline{t_i}$, the k-nearest neurons are chosen as $\overline{w}_1 \ldots \overline{w}_k$.

Table 14: Example of soft k-nearest neighbors classification for a two class problem.

k-Nearest Neurons	$\boldsymbol{W_1^i}$	$\boldsymbol{W_2^i}$
$\overline{w}_1$	0.7543	0.2457
$\overline{w}_2$	0.8311	0.1689
…	…	…
…	0.1392	0.8608
$\overline{w}_k$	0.7543	0.2457
$\sum_{j=1}^{k} W_c^j$	**4.5213**	**1.4787**

The class membership weights of the k neurons are summed. Say the sum of weights is 4.5213 for class 1, and 1.4787 for class 2. The test observation $\overline{t_i}$ is then assigned to class 1. Thus, each neuron provides a test observation its likelihood of belonging to a class of observations, and the sum of the weights provides the test observation an estimate of its likelihood of belonging to each class.

6.3. Experimental Setup and Results

In this section, the data used to obtain the experimental results is first described. The complexities added to the data set are also explained. The setup of the various methods used for comparison are then explained, and the experimental results presented and discussed.

In continuation with previous experiments in this study, the 2SPIRAL data set was used to obtain the experimental results using the developed semi-supervised methodology. 1000 training observations were sampled from the two classes of data. To mimic the scenario where a data set contains only a few labeled observations, 3 different cases were chosen. These three cases represent scenarios where 0.5%, 1%, and 2% of the observations in the training data set are labeled. Thus, for each case, the class labels of 99.5%, 99%, and 98% of the training observations were stripped away. The 3 scenarios thus obtained are shown in Figure 24, with decreasing level of difficulty from left to right. The most difficult scenario is when there are fewest number of training observations are available, making it more difficult for soft labels to be assigned to the unlabeled SOM neurons. The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies.

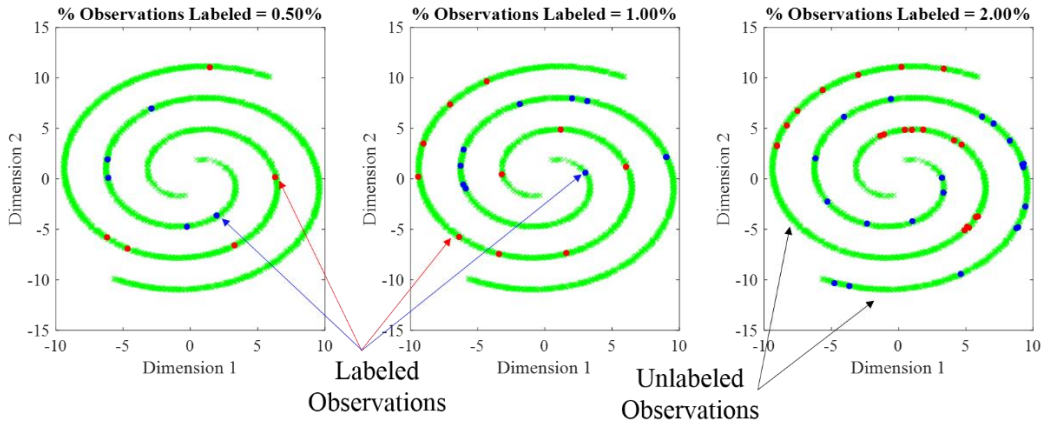

Figure 24: Three different levels of difficulty in the data sets used, obtained by varying percentage of training observations which are labeled.

To validate the effectiveness of the semi-supervised SOM methodology, its performance on the sparsely-labeled 2SPIRAL data set was compared with that of two other classifiers. Following from the comparison throughout this study, SVM-RBF was used as one of the classifiers. The other classifier chosen for comparison was a neural network (NN) with one hidden layer.

The hyperparameters of SVM-RBF were optimized using a built-in MATLAB library which attempts to minimize the cross-validation error by varying SVM's hyperparameters. For the neural network to achieve best performance, the size of its hidden layer was varied from 10 neurons to 100 neurons, and the classification error recorded for each. For each data set, the result corresponding to the hidden layer size yielding the lowest classification error was reported. Additionally, in each case, the

reported test error for these classifiers was the mean test error obtained by training and testing each classifier ten times. The results of this analysis are shown in Figure 25.

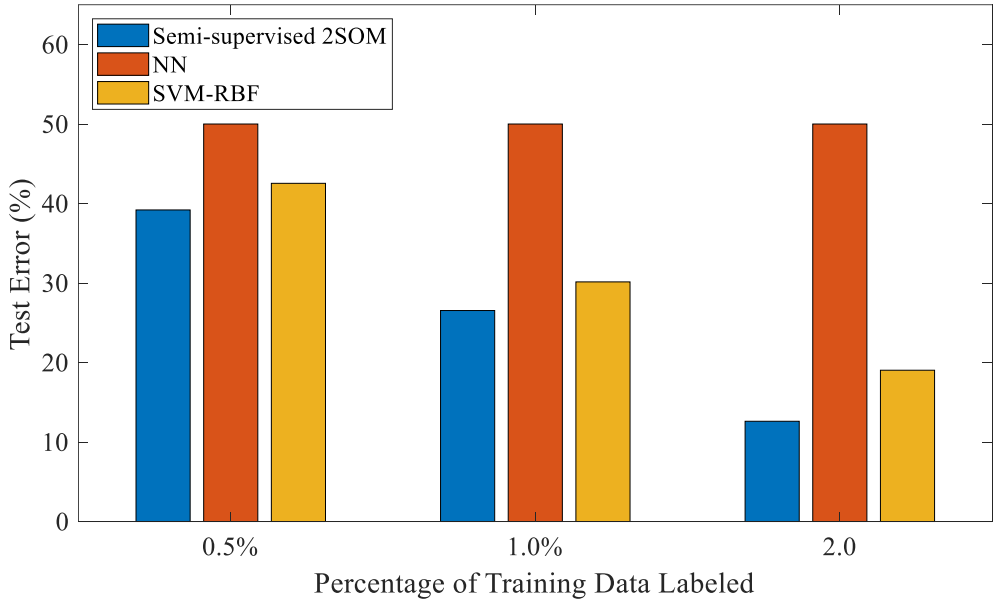

Figure 25: Comparative results of the three methods on sparsely-labeled data sets with different percentage of observations labeled.

In each of the three cases, NN yielded the highest test errors, which was always ~50%. This is the worst performance a classifier can yield since a 50% accuracy is equivalent to a random guess. Interestingly, the performance of NN did not improve as the number of labeled observations in the training data set were increased. It yielded the same test error when 0.5% training observations were labeled as it did when 2% of the training observations were labeled. The performance of SVM-RBF, on the other hand was found to improve as the number of labeled observations in the training data set were increased. While the developed semi-supervised SOM methodology showed the same

trend, its performance was found to be better than that of SVM-RBF by ~3-4% in each case.

6.4. Experiments Using Benchmark Data Sets

In continuation with previous experiments in this study, the four benchmark data sets (CMSC, Iris, Wine, and BCW) were used to obtain the experimental results using the developed semi-supervised methodology. For each data set, 1000 training observations were sampled from the two classes of data. To mimic the scenario where a data set contains only a few labeled observations, 3 different cases were chosen, as in the previous subsection. These three cases represent scenarios where 0.5%, 1%, and 2% of the observations in the training data set are labeled. Thus, for each case, the class labels of 99.5%, 99%, and 98% of the training observations were stripped away. The most difficult scenario is when there are fewest number of training observations are available, making it more difficult for soft labels to be assigned to the unlabeled SOM neurons. The test data set was fixed at 500 observations (250 observations from each class) to obtain unbiased classification accuracies.

To validate the effectiveness of the semi-supervised SOM methodology, its performance on the sparsely-labeled benchmark data sets was compared with that of two other classifiers. Following from the comparison throughout this study, SVM-RBF was used as one of the classifiers. The other classifier chosen for comparison was a neural network (NN) with one hidden layer.

The hyperparameters of SVM-RBF were optimized using a built-in MATLAB library which attempts to minimize the cross-validation error by varying SVM's hyperparameters. For the neural network to achieve best performance, the size of its hidden layer was varied from 10 neurons to 100 neurons, and the classification error recorded for each. For each data set, the result corresponding to the hidden layer size yielding the lowest classification error was reported. Additionally, in each case, the reported test error for these classifiers was the mean test error obtained by training and testing each classifier ten times. The results of this analysis are shown in Figure 26, Figure 27, Figure 28, and Figure 29.

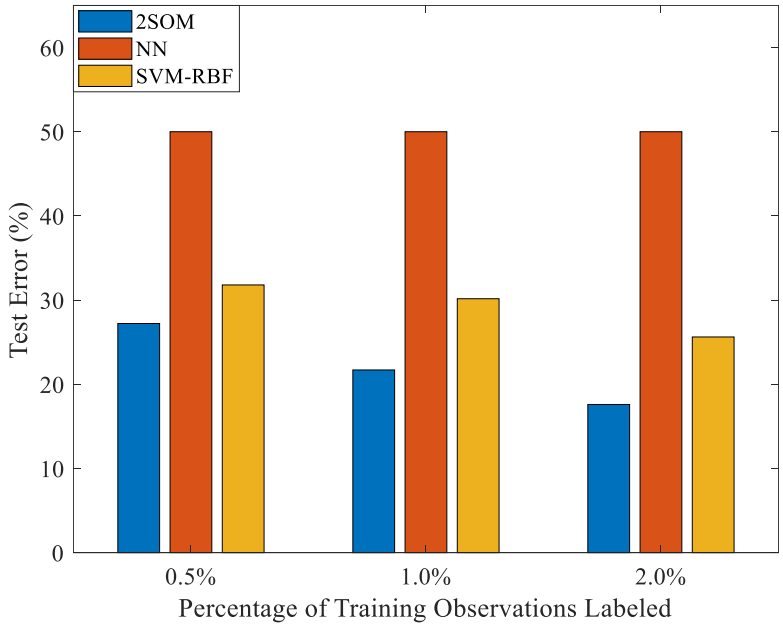

Figure 26: Comparative results of the three methods on sparsely-labeled CMSC data set with different percentage of observations labeled.

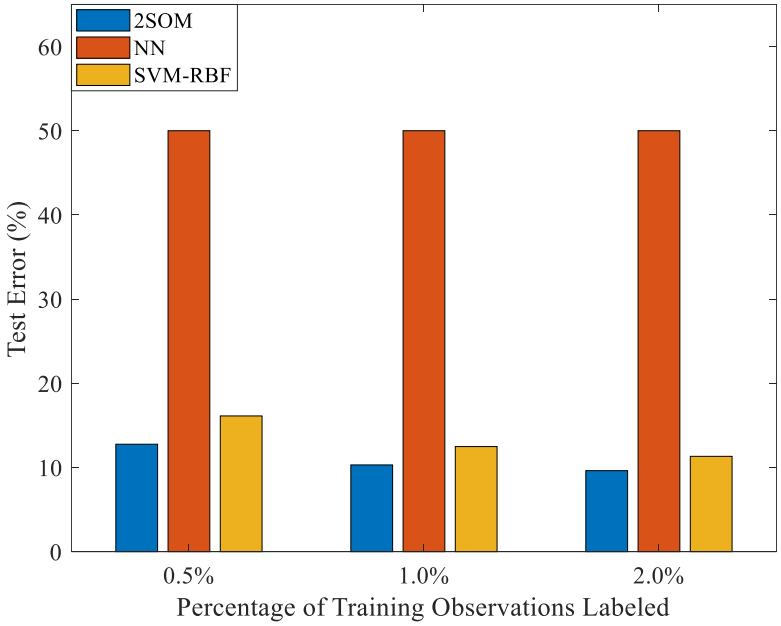

Figure 27: Comparative results of the three methods on sparsely-labeled Iris data set with different percentage of observations labeled.

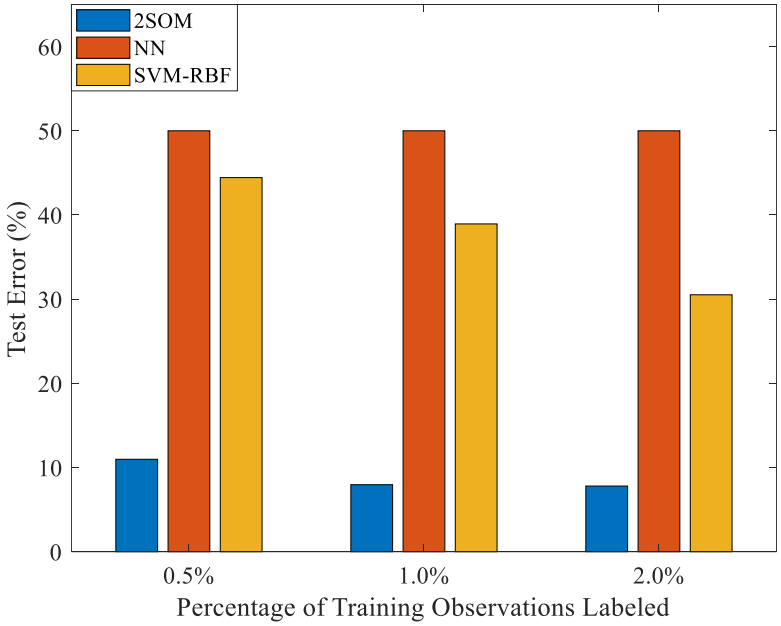

Figure 28: Comparative results of the three methods on sparsely-labeled Wine data set with different percentage of observations labeled.

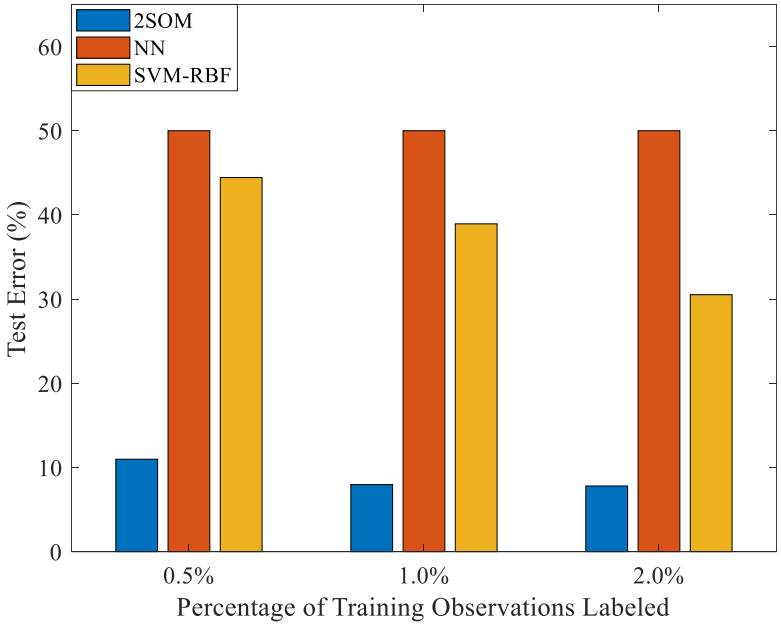

Figure 29: Comparative results of the three methods on sparsely-labeled BCW data set with different percentage of observations labeled.

In case of each of the four benchmark data sets, NN yielded the highest test error, which was always ~50%, regardless of the number of labeled training observations available. As in the case of artificial data sets, the performance of NN did not improve as the number of labeled observations in the training data set were increased. It yielded the same test error when 0.5% training observations were labeled as it did when 2% of the training observations were labeled. The performance of and SVM-RBF, on the other hand was found to improve as the number of labeled observations in the training data set were increased. While the developed semi-supervised SOM methodology showed the same trend, its performance was found to be better than that of SVM-RBF by ~3-

4% in case of CMSC and Iris data sets, and ~20-30% in case of Wine and BCW data sets.

The fact that the SOM methodology performs better than the other two algorithms can be attributed to the additional benefit afforded by semi-supervised learning. The labeling of the SOM neurons using the labeled training data helps the algorithm characterize the class distributions in the data which makes the subsequent classification performance better. What is interesting to note across the results in this section on sparsely-labeled data sets, and the results in the previous section on imbalanced training data sets is the difference in performance of the two algorithms compared. The optimization of hyperparameters for NN and SVM-RBF is the same in both cases. However, in case of imbalanced data, the performance of SVM-RBF was the worst amongst the three methods, indicating that SVM-RBF was prone to poor performance when using imbalanced training data. The performance on NN in that case was better the even the developed cSOM methodology in some cases. In case of sparsely-labeled data, the performance of NN was the worst among the three methods, while SVM –RBF performed better. In both cases, when the level complexity added was the highest, the developed SOM-based methodologies had the best performance. While the performance of different classifiers are prone to different complexities in the training data set, the developed SOM-based methodologies consistently yield good results. This indicates the robustness of the developed methodologies in the face of various complexities encountered in the training data set.

Chapter 7: Contributions

The primary contribution of this work was the development of two types of fault detection methodologies, based on self-organizing maps (SOMs) to specifically tackle condition monitoring data sets riddled with two types of complexities. The first type of complexity, which is frequently encountered in condition monitoring data sets, was class-imbalance. Since systems operate longer in a healthy, rather than faulty state, the volume of healthy condition monitoring data collected from such systems is larger in volume than faulty data. The second type of complexity was that of sparsely-labeled data sets. Since the labeling of raw condition monitoring turns out to be a manual and expensive task, fault detection methodologies must sometimes be developed using only partially-labeled data sets.

Before developing the methodologies to tackle both types of complexities, first, a supervised SOM classifier, called cSOM, was developed. SOM has traditionally been used only in an unsupervised fashion. In this study, a separate SOM was trained on each class of data in the training data set to capture patterns in the training data. Then, the neurons of the trained SOMs were used as prototypes to characterize the relative similarity of test observations from each class of data, based on their Euclidean distance from the nearest neuron in each class-specific SOM.

To deal with the issue of class-imbalance in condition monitoring data sets, the developed cSOM methodology was used. SOMs of equal sizes were trained on the

observations of the imbalanced classes of data in a supervised manner. Since classification using the cSOM methodology only required the prototypical neurons of the training data, the equal-sized SOMs achieved the class balance required to perform correct classification. The size of the SOM required to balance the classes was determined using the genetic algorithm using the classification error, training time, and quantization error as the objective functions. While several methods have been proposed in the literature to deal with class-imbalance, these methods only evaluate the classification error after-the-fact. However, since the end goal of any fault detection methodology is to minimize misclassifications, the classification error was deemed to be an important consideration which must be taken into account while alleviating the imbalance.

To deal with the issue of sparsely-labeled condition monitoring data sets, a two-step transductive semi-supervised methodology was developed. First, a SOM was trained in an unsupervised manner on the entire training data set consisting of both, labeled and unlabeled samples. Based on the proximity of the neurons, to the labeled observations, each neuron was assigned class-membership weights. Finally, to classify previously unseen test observations, the class membership weights of the k neurons closest to the test observations were summed, and the test observation assigned to the class with the highest total weight. Unlike methods presented in the literature, the developed semi-supervised methodology did not require the manual tuning of parameters which can otherwise induce biases in the results.

Chapter 8: Conclusions and Future Work

A novel fault detection methodology, called cSOM, based on supervised self-organizing maps (SOM) was developed, and its use as an efficient algorithm for fault detection was demonstrated. A new health metric, $D_{healthiness}$, which was calculated based on the relative distance of test observations from the patterns in data representing different classes, was introduced. It was shown how this metric can be used to quantify the extent to which a system is in a fault state, and how the developed approach helps overcome pitfalls of the previous SOM-based fault detection method. cSOM's performance was compared with that of six, widely used classifiers. In every case cSOM yielded comparable or better results than the classifiers being compared. The similarities in the operation of 2SOM and SVM-RBF were highlighted, explaining why the two methods consistently yielded comparable classification results. The two methods were also contrasted in the context of their training times and computational complexity. It was shown that 2SOM yields comparable classification performance while saving significant computational resources. These savings were shown to be even more pronounced with increasing sizes of data sets. In general, the following recommendation was made – the decision to use either 2SOM or SVM-RBF for an application is a trade-off between classification accuracy and training time. For large data sets (greater than ~10^7 observations), 2SOM accrued significant savings in training time while losing a few percentage points of classification accuracy over SVM-RBF. The developed method was further generalized to include classification on

multiclass data sets and its effectiveness in performing multiclass classification was demonstrated with its use on the Iris data set.

In case of imbalanced data sets, an optimization-based strategy for cSOM was developed. The genetic algorithm-based optimization made use of classification error, training time and quantization error as the objective functions to be minimized during the process of balancing the data set. While minimizing the quantization error ensured the SOMs trained on each class were most representative of the training data, minimizing the training time ensured that computational resources could be saved in the process. However, the important factor was the use of the classification error as an objective function to be minimized while trying to achieve the balance in the data set. The end goal of any fault detection methodology is always accurate prediction of a fault, and thus using the classification error as an objective function ensured that the end goal was taken into account during the process of balancing, and not after the fact. The developed cSOM methodology was compared against SVM-RBF and shallow neural networks on a variety of data sets with extreme levels of imbalance. The performance of cSOM was found to be the best at the highest levels of imbalance where as few as 10 training observations were available from the minority class of data. It was concluded that even with few training observations, the SOM was able to capture the patterns in the training data efficiently.

In case of sparsely-labeled data sets, a semi-supervised approach was implemented using SOMs. After training a single SOM on all training observations in an

unsupervised manner, the labeled training observations were used to perform soft classification on the prototypical SOM neurons, to assign class-membership weights to them. Classification on test observations was performed using a k-nearest neighbors style approach, classifying test observations to the class with maximum total weight among the k nearest neurons of the test observation. The developed methodology was compared with SVM-RBF and shallow neural networks for data sets with different number of training observations. In every case, the developed methodology yielded the best results. It was concluded that the developed method was efficient at detecting faults even in the face of very sparsely-labeled data sets where only 0.5% of the training observations were labeled.

Future Work

With regards to the developed supervised-SOM classifier, the use of distance metrics (other than Euclidean distance) to compute the metric $D_{healthiness}$ needs to be tested. Specifically, whether the use of a specific distance metric is associated with specific patterns in the training data needs to be investigated.

The imbalance ratio is used to test and compare different classifiers at various levels of class imbalance complexity. However, it was found in this study that given sufficient number of training observations, even at extreme imbalance ratios e.g. 1000:1, classifiers can yield good performance. Whether the parameter imbalance ratio is appropriate to compare classifiers needs to be investigated. A different parameter,

which amalgamates not only the imbalance ratio, but also the number of observations in the imbalanced classes, needs to be developed so that the performance of classifiers can be benchmarked based on this parameter.

Appendix A

SOM Map Size (N)	Size of Neighborhood Function (NF)	Learning Rate (LR)
10x10 (N=100)	1	0.01
10x10 (N=100)	1	0.05
10x10 (N=100)	1	0.1
10x10 (N=100)	2	0.01
10x10 (N=100)	2	0.05
10x10 (N=100)	2	0.1
10x10 (N=100)	3	0.01
10x10 (N=100)	3	0.05
10x10 (N=100)	3	0.1
10x10 (N=100)	4	0.01
10x10 (N=100)	4	0.05
10x10 (N=100)	4	0.1
10x10 (N=100)	5	0.01
10x10 (N=100)	5	0.05
10x10 (N=100)	5	0.1
35x35 (N=1225)	1	0.01
35x35 (N=1225)	1	0.05
35x35 (N=1225)	1	0.1
35x35 (N=1225)	2	0.01
35x35 (N=1225)	2	0.05
35x35 (N=1225)	2	0.1
35x35 (N=1225)	3	0.01
35x35 (N=1225)	3	0.05
35x35 (N=1225)	3	0.1
35x35 (N=1225)	4	0.01
35x35 (N=1225)	4	0.05
35x35 (N=1225)	4	0.1
35x35 (N=1225)	5	0.01
35x35 (N=1225)	5	0.05
35x35 (N=1225)	5	0.1
15x15 (N=225)	1	0.01
15x15 (N=225)	1	0.05
15x15 (N=225)	1	0.1
15x15 (N=225)	2	0.01
15x15 (N=225)	2	0.05
15x15 (N=225)	2	0.1
15x15 (N=225)	3	0.01
15x15 (N=225)	3	0.05
15x15 (N=225)	3	0.1
15x15 (N=225)	4	0.01

15x15 (N=225)	4	0.05
15x15 (N=225)	4	0.1
15x15 (N=225)	5	0.01
15x15 (N=225)	5	0.05
15x15 (N=225)	5	0.1
5x5 (N=25)	1	0.01
5x5 (N=25)	1	0.05
5x5 (N=25)	1	0.1
5x5 (N=25)	2	0.01
5x5 (N=25)	2	0.05
5x5 (N=25)	2	0.1
5x5 (N=25)	3	0.01
5x5 (N=25)	3	0.05
5x5 (N=25)	3	0.1
5x5 (N=25)	4	0.01
5x5 (N=25)	4	0.05
5x5 (N=25)	4	0.1
5x5 (N=25)	5	0.01
5x5 (N=25)	5	0.05
5x5 (N=25)	5	0.1
20x20 (N=400)	1	0.01
20x20 (N=400)	1	0.05
20x20 (N=400)	1	0.1
20x20 (N=400)	2	0.01
20x20 (N=400)	2	0.05
20x20 (N=400)	2	0.1
20x20 (N=400)	3	0.01
20x20 (N=400)	3	0.05
20x20 (N=400)	3	0.1
20x20 (N=400)	4	0.01
20x20 (N=400)	4	0.05
20x20 (N=400)	4	0.1
20x20 (N=400)	5	0.01
20x20 (N=400)	5	0.05
20x20 (N=400)	5	0.1
25x25 (N=625)	1	0.01
25x25 (N=625)	1	0.05
25x25 (N=625)	1	0.1
25x25 (N=625)	2	0.01
25x25 (N=625)	2	0.05
25x25 (N=625)	2	0.1
25x25 (N=625)	3	0.01
25x25 (N=625)	3	0.05
25x25 (N=625)	3	0.1
25x25 (N=625)	4	0.01

25x25 (N=625)	4	0.05
25x25 (N=625)	4	0.1
25x25 (N=625)	5	0.01
25x25 (N=625)	5	0.05
25x25 (N=625)	5	0.1
30x30 (N=900)	1	0.01
30x30 (N=900)	1	0.05
30x30 (N=900)	1	0.1
30x30 (N=900)	2	0.01
30x30 (N=900)	2	0.05
30x30 (N=900)	2	0.1
30x30 (N=900)	3	0.01
30x30 (N=900)	3	0.05
30x30 (N=900)	3	0.1
30x30 (N=900)	4	0.01
30x30 (N=900)	4	0.05
30x30 (N=900)	4	0.1
30x30 (N=900)	5	0.01
30x30 (N=900)	5	0.05
30x30 (N=900)	5	0.1

Appendix B

```
%% Import training data
c = () % c contains the training data with columns as variables and
rows as observations. The last column of c contains labels

%% Define imbalance ratio
ir = ();

%% Partitioning training data into different classes
ind1 = find(c(:,end)==0); %find rows corresponding to class 1
ind2 = find(c(:,end)==1); %find rows corresponding to class 2

c1 = c(ind1,1:end-1);%c1 contains class 0 data (without labels)
c2 = c(ind2,1:end-1);%c2 contains class 1 data (without labels)

train1 = datasample(c1,1000);%sample 1000 observations from class 0
as training data
l1 = zeros(length(train1(:,1)),1);%prepare class 0 training labels
test1  = datasample(c1,250);%sample 250 observations from class 0 as
test data
label1 = zeros(length(test1(:,1)),1);%prepare class 0 test labels

train2 = datasample(c2,round(1000/ir)); %sample observations from
class 1 as training data
%number of observations is determined by the imbalance ratio
l2 = ones(length(train2(:,1)),1); %prepare class 1 training labels
test2  = datasample(c2,250);%sample 250 observations from class 1 as
test data
label2 = ones(length(test2(:,1)),1);%prepare class 1 test labels

%Combining all training data
train = [train1;train2];
label = [l1;l2];
test = [test1;test2];
l_actual = [label1;label2];

%% SOM Training
%Define SOM hyperparameters
neurons      = ();%Specify number of neurons
epochs       = ();%Specify number
neighSize    = ();%Specify neighborhood size
learningRate = ();%Specify learning rate

%Training SOM individually on each class of data
w1 = custom_som(train1,neurons,epochs,neighSize,learningRate);
w2 = custom_som(train2,neurons,epochs,neighSize,learningRate);
%NOTE: same number of neurons are used to train both the majority
and minority class of data thereby achieving balance.
```

```
%% SOM Testing
[dh] = BMUcalculator(test, w1);%computing minimum distance of test
data from class 0 SOM.
[df] = BMUcalculator(test, w2);%computing minimum distance of test
data from class 1 SOM.

ind = find((dh./(dh+df))>0.5);%findinging test observations for
which computed
%normalized distance metric is >0.5

l_predict_som = zeros(length(l_actual),1);
l_predict_som(ind) = 1;%assigning labels to those test observations

%Computing accuracy
True    = length(find(l_actual==l_predict_som));
Total   = length(l_actual);
accSOM = (True/(Total));

%% SOM Training Function
function [weights]=custom_som(data,m,epochs,neighSize,learningRate)
% This function performs SOM training for a given data set and
hyperparameter combination.
% [weights]=custom_som(data,m,epochs,neighSize,learningRate)
% Input:
%     data is the training data without labels data. Every column is
a dimension.
%     m is the number of neurons to use to train the SOM
%     epochs is the number of epochs for which training is performed
%     neighSize is the size of the SOM neighborhood function
%     learningRate is the SOM learning rate
% Output:
%     weights gives the weights of each of the m SOM neurons. The
number of
%     columns in weights is the same as the number of columns in the
%     training data i.e. the dimensionality of the SOM neurons is
the same
%     as that of the training data.
%
%   Rushit Shah, CALCE, University of Maryland, July, 2018.


% Determine the length of each side of the map. A square map is used
by
% default.
lx=sqrt(m);
ly=sqrt(m);

% Train the map
net = selforgmap([lx ly],250,neighSize,'hextop','linkdist');
%default value of coverSteps=100
```

```
net.trainParam.showWindow = 0; % Suppress the training window pop up
net.trainParam.epochs     = epochs; %Set number of epochs
net.trainParam.lr         = learningRate; %Set learning rate.

[net,tr] = train(net,data');
% Calcualte the neuron in the original input data space
neuron=net.iw{1};
weights=neuron;
end


%% BMU Calculator function
function [dist] = BMUcalculator(data,wts)
%Given a set of test observations and SOM neuron weights, this
function
%computes the minimum distance of each test observation from its
closest
%neuron.
%
% [dist]=BMUcalculator(data,wts)
% Input:
%     data is the test data without labels data. Every column is a
dimension.
%     wts contains the weights of each SOM neuron. Every column is a
%     dimension.
%Note: The data and wts must contain the same number of columns.
% Output:
%     dist contains the distance of each observation in data from
its
%     closest neuron in wts.
%
%   Rushit Shah, CALCE, University of Maryland, July, 2018.

    dist  = zeros(length(data(:,1)),1);
    for i=1:length(data(:,1))
        test_vec = data(i,:);
        distance = zeros(length(wts(:,1)),1);
        for j=1:length(wts(:,1))
            distance(j) = sqrt(sum((test_vec - wts(j,:)) .^ 2));
        end
        [dist(i)] = min(distance);
    end
end
```

Appendix C

SOM Map Size (N)	Size of Neighborhood Function (NF)
5x5 (N=25)	1
5x5 (N=25)	2
5x5 (N=25)	3
5x5 (N=25)	4
5x5 (N=25)	5
6x6 (N=36)	1
6x6 (N=36)	2
6x6 (N=36)	3
6x6 (N=36)	4
6x6 (N=36)	5
7x7 (N=49)	1
7x7 (N=49)	2
7x7 (N=49)	3
7x7 (N=49)	4
7x7 (N=49)	5
8x8 (N=64)	1
8x8 (N=64)	2
8x8 (N=64)	3
8x8 (N=64)	4
8x8 (N=64)	5
9x9 (N=81)	1
9x9 (N=81)	2
9x9 (N=81)	3
9x9 (N=81)	4
9x9 (N=81)	5
10x10 (N=100)	1
10x10 (N=100)	2
10x10 (N=100)	3
10x10 (N=100)	4
10x10 (N=100)	5
11x11 (N=121)	1
11x11 (N=121)	2
11x11 (N=121)	3
11x11 (N=121)	4
11x11 (N=121)	5
12x12 (N=144)	1
12x12 (N=144)	2

12x12 (N=144)	3
12x12 (N=144)	4
12x12 (N=144)	5
13x13 (N=169)	1
13x13 (N=169)	2
13x13 (N=169)	3
13x13 (N=169)	4
13x13 (N=169)	5
14x14 (N=196)	1
14x14 (N=196)	2
14x14 (N=196)	3
14x14 (N=196)	4
14x14 (N=196)	5

Appendix D

Sample confusion matrix

		Actual Class	
		0	**1**
Predicted Class	**0**	True Negative	False Negative
	1	False Positive	True Positive

Confusion Matrices for Results Using Artificial Data Sets

1. $d_s = 0.05, IR = 1:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	**1**	**0**	**1**	**0**	**1**
Predicted Class	**0**	244	4	243	18	244	5
	1	6	246	7	232	6	245

2. $d_s = 0.05, IR = 10:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	**1**	**0**	**1**	**0**	**1**
Predicted Class	**0**	242	11	250	23	250	44
	1	8	239	0	227	0	206

3. $d_s = 0.05, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	**1**	**0**	**1**	**0**	**1**
Predicted Class	**0**	185	61	250	250	250	250
	1	65	189	0	0	0	0

4. $d_s = 0.1, IR = 1:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	**1**	**0**	**1**	**0**	**1**
Predicted Class	**0**	236	9	240	11	241	8
	1	14	241	10	239	9	242

5. $d_s = 0.1, IR = 10:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	235	12	250	34	250	112
	1	15	238	0	216	0	238

6. $d_s = 0.1, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	191	57	250	239	250	219
	1	59	193	0	11	0	33

7. $d_s = 0.2, IR = 1:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	235	13	224	35	223	24
	1	15	237	23	215	27	226

8. $d_s = 0.2, IR = 10:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	219	28	249	163	250	170
	1	31	222	1	87	0	80

9. $d_s = 0.2, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	181	70	250	250	250	250
	1	69	180	0	0	0	0

Confusion Matrices for Results Using Benchmark Data Sets

1. $\boldsymbol{CMSC, IR = 1:1}$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	249	4	248	0	250	0
	1	1	246	2	250	0	250

2. $\boldsymbol{CMSC, IR = 10:1}$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	248	7	249	12	250	25
	1	2	243	1	238	0	225

3. $\boldsymbol{CMSC, IR = 100:1}$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	180	66	250	161	250	187
	1	70	184	0	89	0	63

4. $\boldsymbol{Iris, IR = 1:1}$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	249	0	250	0	226	9
	1	1	250	0	250	24	241

5. $\boldsymbol{Iris, IR = 10:1}$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	240	2	250	30	250	59
	1	10	248	0	220	0	191

6. $Iris, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	220	28	250	115	250	197
	1	30	222	0	135	0	53

7. $Wine, IR = 1:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	250	0	250	0	250	0
	1	0	250	0	250	0	250

8. $Wine, IR = 10:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	250	0	250	0	250	31
	1	0	250	0	250	0	219

9. $Wine, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	232	22	250	0	250	182
	1	18	228	0	250	0	68

10. $BCW, IR = 1:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	250	0	250	0	249	0
	1	0	250	0	250	1	250

11. $BCW, IR = 10:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	239	11	248	27	247	1
	1	11	239	2	223	3	249

12. $BCW, IR = 100:1$

		2SOM		NN		SVM-RBF	
		Actual Class		Actual Class		Actual Class	
		0	1	0	1	0	1
Predicted Class	0	210	37	248	60	250	237
	1	40	213	2	190	0	13

Appendix E

cSOM Code for Multiclass Classification

```
%% Import training data
c = () % c contains the c-class training data with columns as
variables and rows as observations. The last column of c contains
labels

%% Partitioning training data into different classes
ind1 = find(c(:,end)==0); %find rows corresponding to class 1
ind2 = find(c(:,end)==1); %find rows corresponding to class 2
.
.
.
indc = find(c(:,end)==c); %find rows corresponding to class c

c1 = c(ind1,1:end-1);%c1 contains class 0 data (without labels)
c2 = c(ind2,1:end-1);%c2 contains class 1 data (without labels)
.
.
.
cc = c(indc,1:end-1);%cc contains class c data (without labels)

train1 = datasample(c1,1000);%sample 1000 observations from class 0
as training data
l1 = zeros(length(train1(:,1)),1);%prepare class 0 training labels
test1  = datasample(c1,250);%sample 250 observations from class 0 as
test data
label1 = zeros(length(test1(:,1)),1);%prepare class 0 test labels

train2 = datasample(c2,1000); %sample observations from class 1 as
training data
l2 = ones(length(train2(:,1)),1); %prepare class 1 training labels
test2  = datasample(c2,250);%sample 250 observations from class 1 as
test data
label2 = ones(length(test2(:,1)),1);%prepare class 1 test labels
.
.
.
trainc = datasample(cc,1000); %sample observations from class 1 as
training data
lc = c*ones(length(train2(:,1)),1); %prepare class c training labels
testc  = datasample(c2,250);%sample 250 observations from class 1 as
test data
labelc = c*ones(length(test2(:,1)),1);%prepare class c test labels
```

```
%Combining all training data
train = [train1;train2;…;trainc];
label = [l1;l2;…,lc];
test = [test1;test2;…;testc];
l_actual = [label1;label2;…;labelc];

%% SOM Training
%Define SOM hyperparameters
neurons      = ();%Specify number of neurons
epochs       = ();%Specify number
neighSize    = ();%Specify neighborhood size
learningRate = ();%Specify learning rate

%Training SOM individually on each class of data
w1 = custom_som(train1,neurons,epochs,neighSize,learningRate);
w2 = custom_som(train2,neurons,epochs,neighSize,learningRate);
.
.
.
wc = custom_som(trainc,neurons,epochs,neighSize,learningRate);


%% SOM Testing
[d0] = BMUcalculator(test, w1);%computing minimum distance of test
data from class 0 SOM.
[d1] = BMUcalculator(test, w2);%computing minimum distance of test
data from class 1 SOM.
.
.
.
[dc] = BMUcalculator(test, wc);%computing minimum distance of test
data from class c SOM.

%Computing metrics for classification
P1=d1./(d1+d2+…+dc);
P2=d2./(d1+d2+…+dc);
.
.
.
Pc=dc./(d1+d2+…+dc);


l_predict_som = zeros(length(l_actual),1);

[max argmax] = max(P1,P2,…,Pc);
l_predict_som = argmax-1;%assigning labels

%Computing accuracy
True    = length(find(l_actual==l_predict_som));
Total   = length(l_actual);
accSOM = (True/(Total));
```

```
%% SOM Training Function
function [weights]=custom_som(data,m,epochs,neighSize,learningRate)
% This function performs SOM training for a given data set and
hyperparameter combination.
% [weights]=custom_som(data,m,epochs,neighSize,learningRate)
% Input:
%     data is the training data without labels data. Every column is
a dimension.
%     m is the number of neurons to use to train the SOM
%     epochs is the number of epochs for which training is performed
%     neighSize is the size of the SOM neighborhood function
%     learningRate is the SOM learning rate
% Output:
%     weights gives the weights of each of the m SOM neurons. The
number of
%     columns in weights is the same as the number of columns in the
%     training data i.e. the dimensionality of the SOM neurons is
the same
%     as that of the training data.
%
%   Rushit Shah, CALCE, University of Maryland, July, 2018.


% Determine the length of each side of the map. A square map is used
by
% default.
lx=sqrt(m);
ly=sqrt(m);

% Train the map
net = selforgmap([lx ly],250,neighSize,'hextop','linkdist');
%default value of coverSteps=100
net.trainParam.showWindow = 0; % Suppress the training window pop up
net.trainParam.epochs     = epochs; %Set number of epochs
net.trainParam.lr         = learningRate; %Set learning rate.

[net,tr] = train(net,data');
% Calcualte the neuron in the original input data space
neuron=net.iw{1};
weights=neuron;
end


%% BMU Calculator function
function [dist] = BMUcalculator(data,wts)
%Given a set of test observations and SOM neuron weights, this
function
%computes the minimum distance of each test observation from its
closest
%neuron.
```

```
%
% [dist]=BMUcalculator(data,wts)
% Input:
%     data is the test data without labels data. Every column is a
dimension.
%     wts contains the weights of each SOM neuron. Every column is a
%     dimension.
%Note: The data and wts must contain the same number of columns.
% Output:
%     dist contains the distance of each observation in data from
its
%     closest neuron in wts.
%
%   Rushit Shah, CALCE, University of Maryland, July, 2018.

    dist  = zeros(length(data(:,1)),1);
    for i=1:length(data(:,1))
        test_vec = data(i,:);
        distance = zeros(length(wts(:,1)),1);
        for j=1:length(wts(:,1))
            distance(j) = sqrt(sum((test_vec - wts(j,:)) .^ 2));
        end
        [dist(i)] = min(distance);
    end
end
```

References

[1] A. K. Jardine, L. Daming and B. Dragan, "A review on machinery diagnostics and prognostics implementing condition-based maintenance.," *Mechanical Systems and Signal Processing,* vol. 20, no. 7, pp. 1483-1510, 2006.

[2] J.-H. Shin and H.-B. Jun, "On condition based maintenance policy," *Journal of Computational Design and Engineering 2,* vol. 2, pp. 119-127, 2015.

[3] C. Chen and M. Pecht, "Prognostics of lithium-ion batteries using model-based and data-driven methods.," in *IEEE Conference on Prognostics and Systems Health Management (PHM),* 2012.

[4] C. Sankavaram, B. Pattipati, A. Kodali, K. Pattipati, M. Azam, S. Kumar and M. Pecht, "Model-based and data-driven prognosis of automotive and electronic systems.," in *IEEE International Conference on Automation Science and Engineering (CASE),* 2009.

[5] H. Qiao, Z. He, Z. Zhang and Y. Zi, "Fault diagnosis of rotating machinery based on improved wavelet package transform and SVMs ensemble.," *Mechanical Systems and Signal Processing,* vol. 21, no. 2, pp. 688-705, 2007.

[6] V. Every, P. Michael, M. Rodriguez, B. C. Jones, A. A. Mammoli and M. Martinez-Ramon, "Advanced detection of HVAC faults using unsupervised SVM novelty detection and Gaussian process models.," *Energy and Buildings,* vol. 147, pp. 216-224, 2017.

[7] Y. Jie and J. S. Qin, "Multimode process monitoring with Bayesian inference-based finite Gaussian mixture models.," *AlChE Journal,* vol. 54, no. 7, pp. 1811-1829, 2008.

[8] Y. Jie and J. S. Qin, "Multiway Gaussian mixture model based multiphase batch process monitoring.," *Industrial & Engineering Chemistry Research,* vol. 48, no. 18, pp. 8585-8594, 2009.

[9] B. Y. Vyas, B. Das and R. Prakash, "Improved fault classification in series compensated trasnmission line: comparative evaluation of Chebyshev neural network training algorithms.," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 27, no. 8, pp. 1631-1642, 2016.

[10] M. Bach-Andersen, B. Romer-Odgaard and O. Winther, "Deep learning for automated drivetrain fault detection.," *Wind Energy,* vol. 21, no. 1, pp. 29-41, 2018.

[11] R. Hao, Y. Chai, J. Qu, X. Ye and Q. Tang, "A novel adaptive fault detection methodology for complex system using deep belief networks and multiple models: A case study on cryogenic propellant loading system.," *Neurocomputing,* vol. 275, pp. 2111-2125, 2018.

[12] K. Worden, W. J. Staszewski and J. J. Hensman, "Natual computing for mechanical systems research: A tutorial overview.," *Mechanical Systems and Signal Processing,* vol. 25, pp. 4-111, 2011.

[13] V. Lopez, A. Fernandez, S. Garcia, V. Palade and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current rends on using data intrinsic characteristics.," *Information Sciences,* vol. 250, pp. 113-141, 2013.

[14] O. Loyola-Gonzalez, J. F. Martinez-Trinidad, J. A. Carrasco-Ochoa and M. Garcia-Borroto, "Study of the impact of resampling methods for contrast pattern based classifiers in imbalanced databases.," *Neurocomputing,* vol. 175, pp. 9355-947, 2016.

[15] G. M. Weiss, "Mining with rarity: A unifying framework.," *ACM SIGKDD Explorations Newsletter,* vol. 6, no. 1, pp. 7-19, 2004.

[16] B. Cigdem and R. Fisher, "Classifying imbalanced data sets using similarity based hierarchical decomposition.," *Pattern Recognition,* vol. 48, no. 5, pp. 1653-1672, 2015.

[17] M. Wasikowski and X. W. Chen, "Combating the small sample class imbalance problem using feature selection.," *IEEE Transactions on Knowledge and Data Engineering,* vol. 22, no. 10, pp. 1388-1400, 2010.

[18] I. Marti-Diaz, D. Morinigo-Sotelo, O. Duque-Perez and R. d. J. Romero-Troncoso, "Early fault detection in induction motors using AdaBoost with imbalanced small data and optimized sampling.," *IEEE Transaction on Industry Applications,* vol. 53, no. 3, pp. 3066-3075, 2017.

[19] P. Potocnik and E. Govekar, "Semi-supervised vibration-based classification and condition monitoring of compressors.," *Mechanical Systems and Signal Processing,* vol. 93, pp. 51-65, 2017.

[20] A. Stanescu, K. Tangirala and D. Caragea, "Study of transductive learning and unsupervised feature construction methods for biological sequence classification.," in *Proceedinngs of the 2016 IEEE/ACM International Conference on Advances in Social Network Analysis and Mining,* 2016.

[21] J. Yuan and X. Liu, "Semi-supervised learning and condition fusion for fault diagnosis.," *Mechanical Systems and Signal Processing,* vol. 38, no. 2, pp. 615-627, 2013.

[22] R. N. Shah, M. H. Azarian, J. E. Wedgwood, M. Krein and M. Pecht, "Fault detection using a multiple self-organizing map methodology.," *Mechanical Systems and Signal Processing (Submitted),* 2018.

[23] L. Duan, M. Xie, T. Bai and J. Wang, "A new support vector data description method for machinery fault diagnosis with unbalanced data sets.," *Expert Systems with Applications,* vol. 64, pp. 239-246, 2016.

[24] M. Wentao, L. He, Y. Yan and J. Wang, "Online sequential prediction of bearings imbalanced fault diagnosis by extreme learning machine.," *Mechanical Systems and Signal Processing,* vol. 83, pp. 450-473, 2017.

[25] R. Razavi-Far, P. Baraldi and E. Zio, "Dynamic weighting ensembles for incremental learning and diagnosing new concept class faults in nuclear power systems.," *IEEE Transactions on Nuclear Science,* vol. 59, no. 5, pp. 2520-2530, 2012.

[26] X. Jin, F. Yuan, T. W. S. Chow and M. Zhao, "Weighted local and global regressive mapping; A new manifold learning method for machine fault classification.," *Engineering Applications of Artificial Intelligence,* vol. 30, pp. 118-128, 2014.

[27] X. Xhang, B. Wang and X. Chen, "Intelligent fault diagnosis of roller bearings with multivariable ensemble-based incremental support vector machine.," *Knowledge-Based Systems,* vol. 89, pp. 56-85, 2015.

[28] S. Cateni, V. Colla and M. Vannucci, "A method for resampling imbalanced data sets in binary classification tasks for real-world problems.," *Neurocomputing,* vol. 135, pp. 32-41, 2014.

[29] F. De Morsier, D. Tuia, M. Borgeaud, V. Gass and J.-P. Thiran, "Semi-supervised novelty detection using SVM entire solution path.," *IEEE Transactions on Geoscience and Remote Sensing,* vol. 51, no. 4, pp. 1939-1950, 2013.

[30] Z.-J. Zha, T. Mei, J. Wang, Z. Wang and X.-S. Hua, "Graph-based semi-supervised learning with multiple labels.," *Journal of Visual Communication and Image Representation,* vol. 20, no. 2, pp. 97-103, 2009.

[31] X. Wang, H. Feng and Y. Fan, "Fault detection and classification for complex processes using semi-supervised learning algorithm.," *Chemometrics and Intelligent Laboratory Systems,* vol. 149, pp. 24-32, 2015.

[32] H.-C. Yan, J.-H. Zhou and C. K. Pang, "Gaussian mixture model using semisupervised learning for probabilistic fault diagnosis under new data categories.," *IEEE Transactions on Instrumentation and Measurement,* vol. 66, no. 4, pp. 723-733, 2017.

[33] J. Chen, X. Zhang, N. Zhang and K. Guo, "Fault detection for turbine engine disk using adaptive Gaussian mixture model.," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering,* vol. 231, no. 10, pp. 827-835, 2017.

[34] G. Albaei, A. Selamat and H. Fujita, "An emipirical study based on semi-supervised hybrid self-organizing map for software fault prediction.," *Knowledge-Based Systems,* vol. 74, pp. 28-39, 2015.

[35] E. Fix and J. L. Hodges Jr., "Discriminatory analysis - nonparametric discrimination: Consistency properties.," USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[36] T. Kohonen, "The self-organizing map," *Neurocomputing,* vol. 21, no. 1, pp. 1-6, 1998.

[37] I. Valova, G. Georgiev and N. Gueorguieva, "Initialization issues in self-organizing maps," *Procedia Computer Science,* vol. 20, pp. 52-57, 2013.

[38] H. Qiu, Lee, Jay, J. Lin and G. Yu, "Robust performance degradation assessment methods for enhanced rolling element bearing prognostics.," *Advanced Engineering Informatics,* vol. 17, no. 3, pp. 127-140, 2003.

[39] M. Lichman, *UCI Machine Learning Repository,* University of California, Irvine, School of Information and Computer Sciences, 2013.

[40] K. P. Bennett and E. J. Bredensteiner, "Duality and geometry in SVM classifiers.," in *International Conference on Machine Learning*, San Francisco,CA, USA, 2000.

[41] H. Yu, J. Yang and J. Han, "Classifying large data sets using SVMs with hierarchical clusters.," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* , 2003.

[42] P. Liang, W. Li, D. Liu and J. Hu, "Large-scale image classification using fast SVM with deep quasi-linear kernel.," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[43] T. Lee, K. B. Lee and C. O. Kim, "Performance of machine learning algorithms for class-imblanced process fault detection problems.," *IEEE Transactions on Semiconductor Manufacturing,* vol. 29, no. 4, pp. 436-445, 2016.

[44] A. Rehan, S. Kwek and N. Japkowicz, "Applying support vector machines to imbalanced datasets.," in *European Conference on Machine Learning*, Beidelberg, 2004.

[45] W. Gang and E. Y. Chang, "Adaptive feature-space conformal transformation for imbalanced-data learning," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003.

[46] K. Veropoulos, C. Campbell and N. Cristianini, "Controlling the sensitivity of support vector machines.," in *Proceedings of the International Joint Conference on AI,* 1999.

[47] T. Kohonen, MATLAB implementations and applications of the self-organizing map., Helsinki, Finland: Unigrafia Oy, 2014.

[48] J. H. Holland, "Genetic algorithms.," *Scientific American,* vol. 267, no. 1, pp. 66-73, 1992.

[49] X. Zhu, Z. Ghahramani and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions.," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003.

[50] D. Zhou, O. Bousquet, T. N. Lal, J. Weston and B. Scholkopf, "Learning with local and global consistency.," in *Advances in Neural Information Processing Systems,* 2004.

[51] M. Zhao, B. Li, J. Qi and Y. Ding, "Semi-supervised classification for rolling fault diagnosis via robust sparse and low-rank model.," in *IEEE 15th International Conference on Industrial Informatics (INDIN) 2017*, 2017.