FAULT DETECTION AND IDENTIFICATION OF VEHICLE STARTERS AND ALTERNATORS USING MACHINE LEARNING TECHNIQUES

FAULT DETECTION AND IDENTIFICATION OF VEHICLE STARTERS AND ALTERNATORS USING MACHINE LEARNING TECHNIQUES

BY

ESSAM SEDDIK

A THESIS

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from McMaster University, I agree that the Libraries of this University may make it freely available for inspection. I further agree that the permission for copying this thesis in any manner, in whole or in part for scholarly purposes, may be granted by the professors who supervised my thesis work or, in their absence, by the Head of the department or the Faculty Dean in which my thesis work was conducted. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and McMaster University in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis, in whole or in part, should be addressed to:

Head of the Department of Mechanical Engineering

McMaster University

Faculty of Engineering

1280 Main Street West

Hamilton, Ontario L8S 4L6, Canada.

Master of Applied Science (2015) McMaster University
(Mechanical Engineering) Hamilton, Ontario, Canada

TITLE: Fault Detection and Identification of Vehicle Starters and Alternators Using Machine Learning Techniques

AUTHOR: Essam Seddik

SUPERVISOR: Professor Saeid Habibi

NUMBER OF PAGES: xvi, 169

Abstract

Cost reduction is one of the main concerns in industry. Companies invest considerably for better performance in end-of-line fault diagnosis systems. A common strategy is to use data obtained from existing instrumentation. This research investigates the challenge of learning from historical data that have already been collected by companies. Machine learning is basically one of the most common and powerful techniques of artificial intelligence that can learn from data and identify fault features with no need for human interaction. In this research, labeled sound and vibration measurements are processed into fault signatures for vehicle starter motors and alternators. A fault detection and identification system has been developed to identify fault types for end-of-line testing of motors.

However, labels are relatively difficult to obtain, expensive, time consuming and require experienced humans, while unlabeled samples needs less effort to collect. Thus, learning from unlabeled data together with the guidance of few labels would be a better solution. Furthermore, in this research, learning from unlabeled data with absolutely no human intervention is also implemented and discussed as well.

"My work is dedicated to my family members in Canada, who provided me with love and support, and of course my parents to whom I dedicate every single achievement in my life"

Acknowledgements

The author would like to express his gratitude to his supervisor Dr. S. R. Habibi, for his supervision, advice, guidance and encouragement from the very early stage of this research until the writing phase of this thesis.

Financial support provided by the School of Graduate Studies, the Department of Mechanical Engineering.

I convey special acknowledgements to my lab colleagues who were always there for help since my first day at CMHT, especially Mahmoud Ismail who provided me with the output of his M.Sc. thesis to start with and move on to my research goals.

I would like to thank also Dr. Mohamed Al-Ani for his advice and guidance during the first year and Cam Fisher for his technical support and for providing me with research facilities.

I gratefully thank my lovely fiancée for believing in me, her support, love and patience.

No words can describe my gratitude to my parents who always give me endless support, love and encouragement.

Table of Contents

List of Figures

List of Tables

Abbreviations

AE	Auto-Encoder
ANN	Artificial Neural Network
BM	Boltzmann Machines
DBN	Deep Belief Network
DL	Deep Learning
EM	Expectation Maximization
EMSPCA	Extended Multi-Scale Principle Component Analysis
FCM	Fuzzy C-means
FDD	Fault Diagnosis and Detection
FM	Feature Map
IEMSPCA	Industrial Extended Multi-Scale Principle Component Analysis
LP	Label Propagation
ML	Machine Learning
PCA	Principle Component Analysis
RBM	Restricted Boltzmann Machine
SAE	Stacked Auto-Encoders
SVM	Support Vector Machine
WTP	Wavelet Transformation Packet

Chapter 1

Introduction

1.1 Overview

The human brain is an example of a very complex network. Learning is one of the unique capabilities that our brains have. All humans continuously learn from their surrounding environments. Some need a teacher to guide them and give them information and examples. Others deal with problems on their own. People generally complement what they have already learned and gain experience from situations they go through. Some go through problems, gain experience from their actions and solve problems without having prior examples or hints. [1]

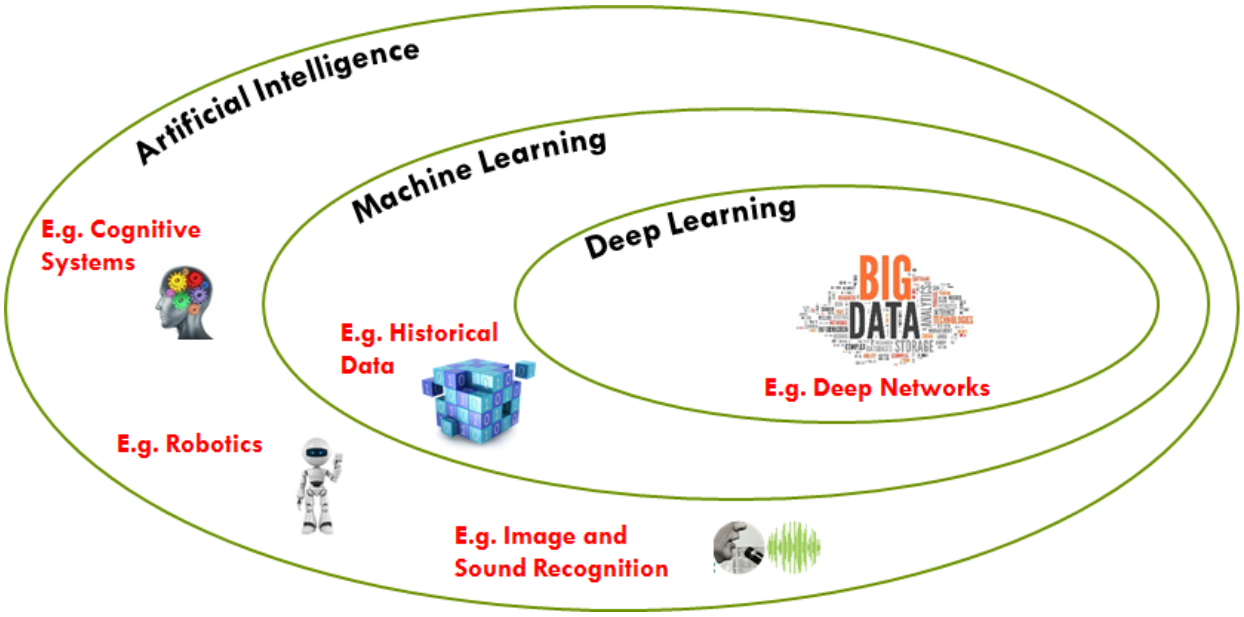


Figure (1.1) Hierarchy of Artificial Intelligence

Scientists have attempted to mimic the human brain intelligence through what is called "Artificial Intelligence (AI)" using software. The Venn diagram in figure (1.1) shows the relation between different topics and examples from AI. Machine learning, which is a sub-category of representation learning, is one of the most popular approaches in AI. It will be discussed in details through this project. [1]

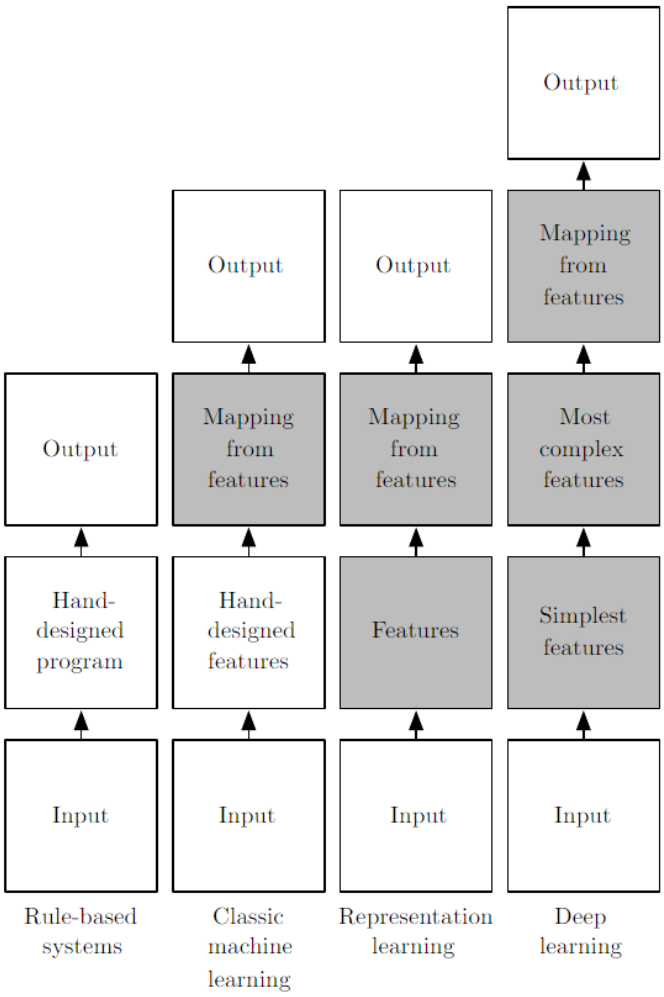


Figure (1.2) Hierarchy of different Learning Techniques [2]

Figure (1.2) represents four different techniques typically used under artificial intelligence. The figure shows how they relate to one another and with other AI disciplines. Items in shaded boxes are those that are able to learn. Applications of machine learning are countless. Image recognition, speech recognition and speaker identification are the most popular examples. Figure (1.3) shows an example of a machine learning model using neural networks with 3 hidden layers. Raw sensory input data, caught by a regular camera, is not easy for a computer to understand. The example shows an image

represented as a collection of pixel values. Mapping from this set of pixels to an object identity is very complicated. Deep learning makes it easier by breaking the desired mapping into a series of nested mappings, each mapping capturing a feature layer of the model. The operation that occurs between the different levels is called **sparse coding**, which calculates the probability for each pixel of a feature. It aims to find an over-complete set of basis vectors to represent an input vector as a linear combination of these basis vectors. This idea of hierarchical learning can be used in many other applications. For example, in speaker identification, the speech could be recorded using a microphone, then the voice is analyzed using power spectrums to catch the inner features (a.k.a. phonemes) and then used to build a hierarchy of features until the voice of the speaker is identified.

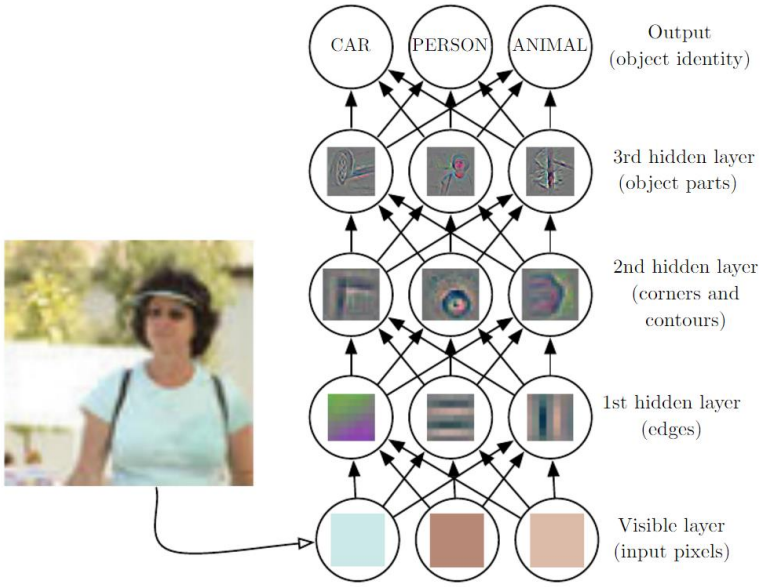


Figure (1.3) Example of Machine learning for image recognition

However, feature detection, extraction and identification is the goal of this project as the input data for our work is a set of data processed into features referred to as fault signatures as outlined in [2] using Industrial Extended Muti-scale Principle Components Analysis (IEMSPCA) method. Different

types of machine learning algorithms will be applied for detecting and identifying faults in starter motors and alternators.

1.2 Research Motivation

This project aims to develop a fault detection and identification software for starter motors and alternators with absolutely no need for human interaction. The software would learn from labeled historical data and would detect and diagnose faults of new motors with higher detection rate than humans. However, the big challenge is to let it learn from unlabeled data as well and be capable to generalize rules on its own with the highest possible detection rate.

1.3 Research Objectives

The main objective of this research is to create a machine learning algorithm for vehicle starters and alternators that is able to do the following tasks:

- Learn from labeled data.
- Learn from both labeled and unlabeled data.
- Learn from only unlabeled data and generalize.
- Detect and identify known fault conditions.
- Detects new previously unknown faults.

The indicator for assessing the software is its effectiveness in correctly detecting and diagnosing fault conditions.

1.4 Organisation of the thesis

This thesis is organised as follows. Chapter 2 reviews the main literature on machine learning techniques. It starts by reviewing neural networks and expands up to deep learning algorithms. Supervised, semi-supervised and unsupervised learning algorithms are considered. In chapter 3, a supervised learning model is presented and discussed in details. Two learning algorithms are proposed and applied to artificial neural networks (ANN) and support vector machines. The performance of the algorithms are presented and compared.

Chapter 4 develops a semi-supervised learning model. A graph-based learning algorithm, referred to as Label Propagation (LP), is used for training. A completely unsupervised learning model is proposed in chapter 5 in order to map unlabeled data. Chapter 4 and 5 also provide performance comparison of semi-supervised and unsupervised learning models to that of the supervised model of chapter 3. Concluding remarks and outcomes are discussed in chapter 6. Recommendations for future research are also provided.

Chapter 2

Literature Review

2.1 Types of Learning

Two important forms of learning are: Learning with a teacher and learning without a teacher (according to terminologies introduced in [3]).

2.1.1 Learning with a Teacher

Electronically called supervised learning, means having prior knowledge about the system or the problem that the machine is to learn or resolve [3]. This knowledge most commonly comes in a form of a set of input-output examples. Similarly to how humans learn, a teacher feeds us with knowledge, experiences and solutions to situations that we may encounter. Through this training, we get the ability to solve problems based on what was taught. Figure (2.1) shows a block diagram of supervised learning. The figure describes the state of the environment as a training vector. This vector feeds both

the teacher and the learning system as well. The network parameters of the learning system are adjusted by the error signal which is actually the difference between the desired response and the actual response of the system. A closed-loop feedback is effectively created for error correction purposes. Only the environment is exempted from this loop as it is always unknown and changes over time. The true error is averaged over all possible input-output examples.

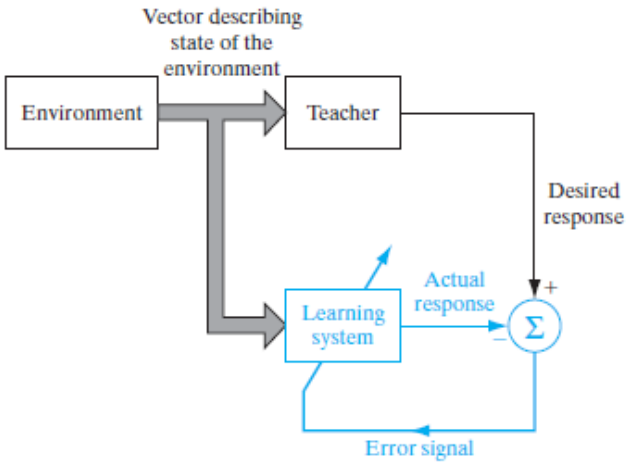


Figure (2.1) Learning with a teacher [3]

2.1.2 Learning without a teacher

The second type of learning is basically the learning process with no teacher. Thus, no input-output examples are provided. It is difficult indeed even for a human being whose brain is much more complex than neural networks to deal with something without having prior information [3]. This challenge requires a kind of smartness that allows the system to generate concepts and rules based on observations over the features of the data being used for learning. These observations help with comparisons between the patterns in different input samples. So the task will be input-output mapping

instead of input-output examples. Two subcategories can be identified under learning without a teacher: Reinforcement learning and unsupervised learning.

2.1.2.1 Reinforcement learning

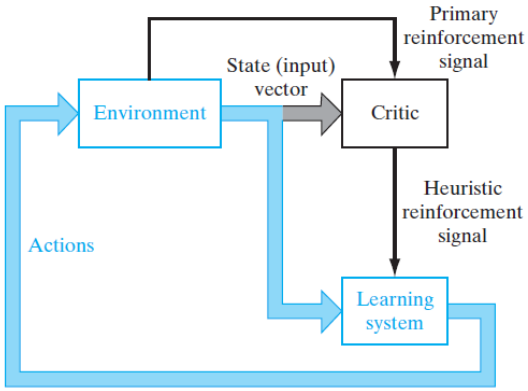


Figure (2.2) Reinforcement Learning [3]

Figure (2.2) shows the block diagram of one form of reinforcement learning. Learning an input-output mapping is in continuous interaction with the environment. A critic is mainly a signal corrector that receives feedback from the environment. It converts the primary reinforcement signal into a higher quality reinforcement signal called the heuristic reinforcement signal. This signal feeds the learning system which in turn observes a temporal sequence of stimuli received from the environment eventually resulting in the generation of a high quality signal. That is what is called delayed reinforcement learning and, it is difficult to perform mainly because no desired response is provided by a teacher [3].

2.1.2.2 Self-organized Learning

Also known as unsupervised learning, the system learns only and directly from the environment [3]. There is no teacher nor critic to help out, that is why it is known as self-organized learning. The input data gets clamped to the input layer without any labels. Then the system comes up with learning rules and a serious competition starts between the neurons of the internal layers for the opportunity to respond to the features contained in the input data. A winner-takes-all strategy occurs between the neurons as the one having the greatest total, wins the competition and turns on, showing the suitable label for each input. Figure (2.3) shows the block diagram for unsupervised learning [3].

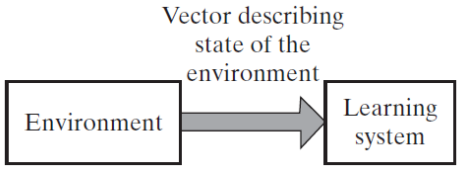


Figure (2.3) Self-Organised Learning [3]

2.2 Neural Networks

In cognitive science, neural networks are the central nervous systems of the human brain. However, in machine learning, artificial neural networks (ANNs) are a set of models inspired by biological neural networks that are used to estimate functions given a large number of inputs [4]. ANNs consist of a number of "neurons" which build interconnected layers with numeric tunable weights to make the network adaptive to inputs and capable of learning. Since a wide variety of tasks was hard to solve with ordinary rule-based programming, neural networks have been invented to tackle bioengineering problems such as computer vision and speech recognition. Let us start from the smallest unit of a typical neural network: The perceptron. [5]

2.2.1 The Perceptron

The basic neural network building block is the perceptron, which is one of the earliest supervised learning algorithms. The idea started when researchers had a number of data points, some of them from similar objects and others from different objects (for example "car" and "not car"). The challenge was to identify a label for a new data point. A very basic idea was to look at the closest neighbor and assume that it belongs to the same label. As shown in Figure (2.4), this idea was implemented by separating line that would divide the data into two classes. Any point that is above the line would be a car and those who are below would be something else. This actually means that this line is working as a classifier. [6,7]

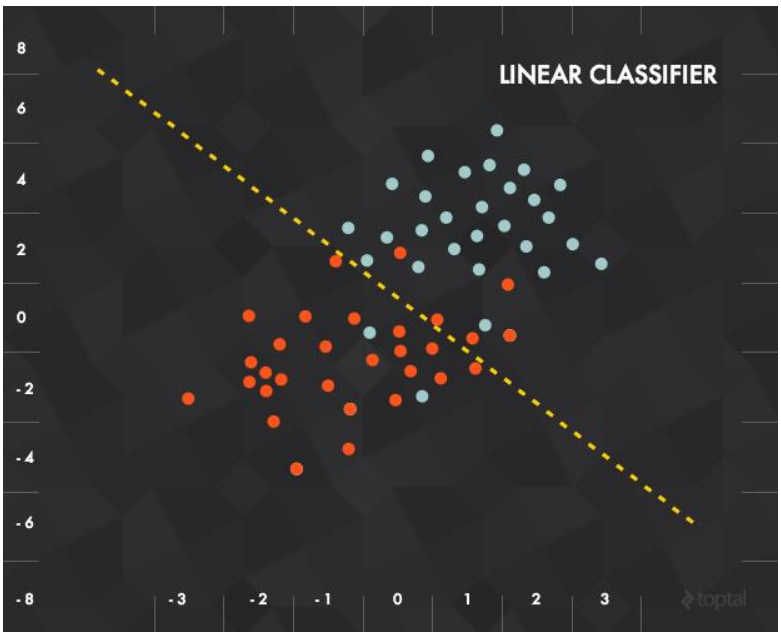


Figure (2.4) Linear Classifier [7]

This line is represented mathematically as a weighted sum transfer function f(x). It is a sum of combinations of input vectors (x) and their weights (w) with a bias (b) as follows:

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \tag{2.1}$$

This transfer function is actually the value of the threshold that is fed into an activation function h(x), illustrated in figure (2.5), which works as a threshold cut-off. If the result of this function is greater than a fixed value, the activation will notify that it is a "1" (i.e. car), otherwise, a "0" will show up (i.e. not a car)[7].

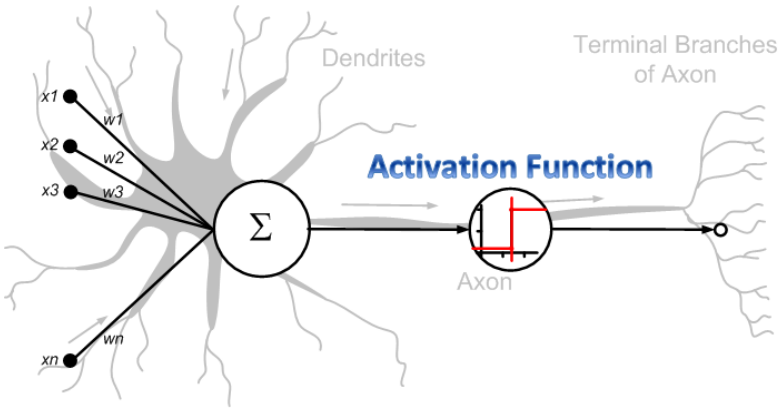


Figure (2.5) Activation Function

$$h(\mathbf{x}) = \begin{cases} 1 : & \text{if } f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 : & \text{otherwise} \end{cases} \tag{2.2}$$

The main job of the weights (w) in the transfer function is to minimize the output error, which is basically the difference between the desired and the actual output. Several error functions can be used for calculating the output error such as Mean Square Error and Least Square Error. A single perceptron is simple and easy to apply, but unfortunately it can only learn linearly separable functions [6,7]. This is considered as a major drawback because some simple functions such as XOR cannot be classified using a single perceptron as shown in figure (2.6).

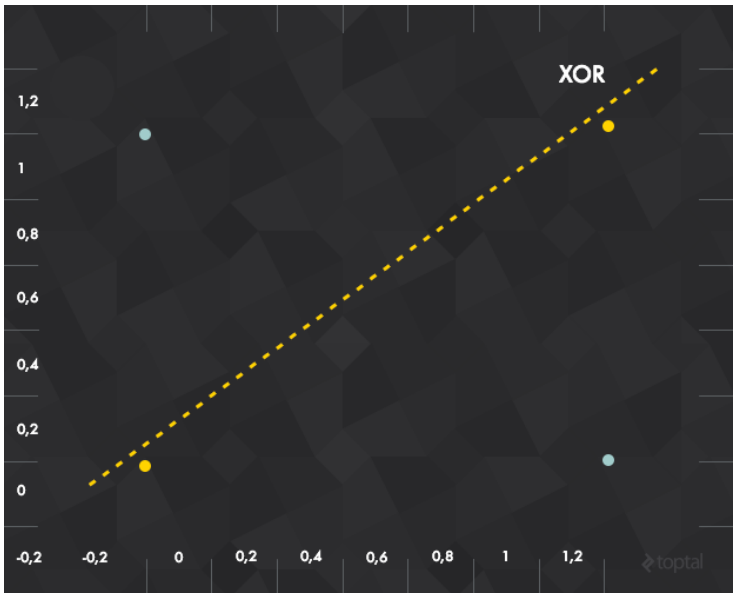


Figure (2.6) XOR Problem [7]

This disadvantage leads us to use a multilayered perceptron, which is also known as a "Feedforward neural network".

2.2.2 Neural Network Structure

The basic construction of a neural network is a number of layers, each layer has a number of perceptrons. Each perceptron can be called a "neuron" or a "unit" within a neural network interchangeably. The network consists of one input layer, one output layer and at least one hidden layer. The neural network shown in figure (2.7) has a 3-unit input layer, a 4-unit hidden layer and a 2-unit output layer. Connections between neurons have weights (w) similar to those of the perceptron weights. Each unit in a layer (n) is typically connected to every unit of the previous layer (n-1). These connections can be disconnected simply by setting their weight to zero.

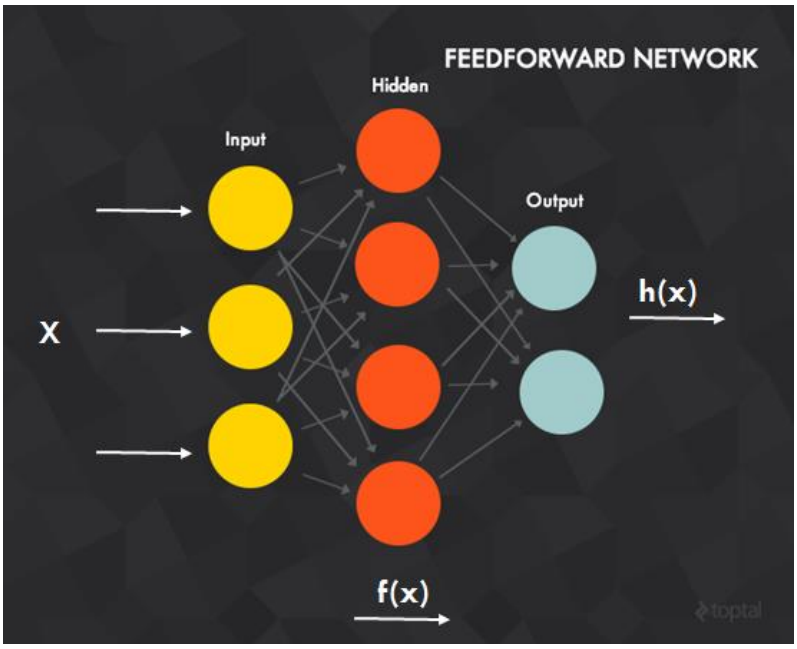


Figure (2.7) FeedForward Neural Network [7]

First, the input vector is fed to the input layer. The output vector from each layer forms the input to the next. This input vector is propagated forward to the hidden layer. The result of the output layer is the output of the whole network [6,7].

A linear composition of a bunch of linear functions is still just a linear function. Thus, in the case of using a linear activation function, the feedforward neural network is no more powerful than the perceptron, no matter how many layers it has. That is why most neural networks use non-linear activation functions. A single hidden layer is not always powerful enough to learn functions. That said, we often learn better in practice with multiple hidden layers [6,7]. Figure (2.8) shows the different activation functions that can be used in neural networks.

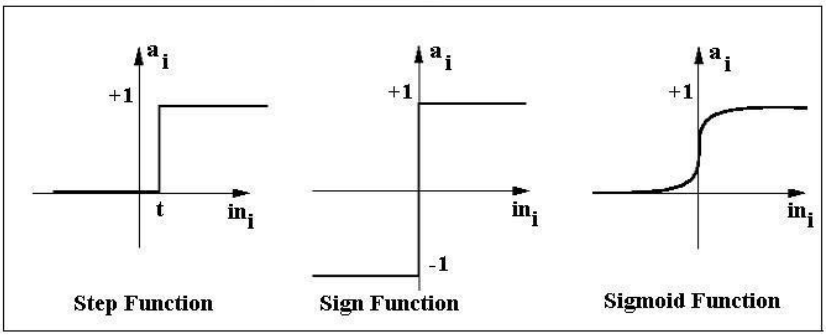


Figure (2.8) Different Activation Functions

A feed forward neural network is not efficient enough to give best results with only one trial. More trials do not mean only repetition, but they also need applying corrections to the synaptic weights that change over time during the training process. This job can be performed by a backpropagation algorithm. Most introductory machine learning classes tend to stop with feedforward neural networks, but the space of possible network architectures is far richer [6].

2.2.3 Backpropagation algorithm

The most common algorithm for supervised training of the multi-layer perceptron (MLP) in machine learning is back propagation. It is one of the simplest and most effective algorithms to implement. The term "back propagation" showed up first by Henry J. Kelly in 1960 and Arthur E. Bryson in 1961 [13]. It became popular after the publication of a seminal book titled Parallel Distributed Processing by Rumelhart and McClelland in 1986. Before going through the mathematical representation of the backpropagation algorithm, it is important to know about the phases that form this algorithm. There are two main phases to a backpropagation algorithm [8]: Forward phase and Back phase.

During the forward phase, the weights of the connections of the network do not change and the input signal is propagated through the network passing by all layers until it reaches the output layer. Meanwhile, the activation potentials and outputs of the neurons in the network are being calculated. On the way back, the output error is calculated as the difference between the actual result of the whole network and the desired response. The resulting error signal is propagated back through all the layers of the network in the opposite direction. During this phase, the synaptic weights of the network are updated and adjusted. Adjustments to the output layer is straightforward and much easier than for hidden layers. The correction $\Delta w_j(n)$ applied to the synaptic weight connecting neuron *i* to neuron *j* is defined by the delta rule [6]:

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning\text{-} \\ rate\ parameter \\ \eta \end{pmatrix} \times \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} input\ signal \\ of\ neuron\ j, \\ y_i(n) \end{pmatrix} \tag{2.3}$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron *j* is an output node or a hidden node. If neuron *j* is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi'_j(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron *j* as follows:

$$\delta_j(n) = e_j(n)\, \varphi'_j(v_j(n)) \tag{2.4}$$

If neuron *j* is a hidden node, $\delta_j(n)$ equals the product of the associated derivative $\varphi'_j(v_j(n))$ and the weighted sum of the δ_s computed for the neurons in the next hidden or output layer that are connected to neuron *j*:

$$\delta_j(n) = \varphi'_j(v_j(n))\, \Sigma_k\, \delta_k(n)\, w_{kj}(n) \tag{2.5}$$

Where neuron k is an output node [3]. The most preferred method of on-line implementation of the backpropagation algorithm is the stochastic gradient method. Figure (2.9) represents this method graphically.

The optimal value for each weight is that at which the error achieves a global minimum. The weights are updated during the training phase in small steps in such a way that they are always trying to reach the global minimum. But practically, it is a difficult task as you often end up in local minima [6].

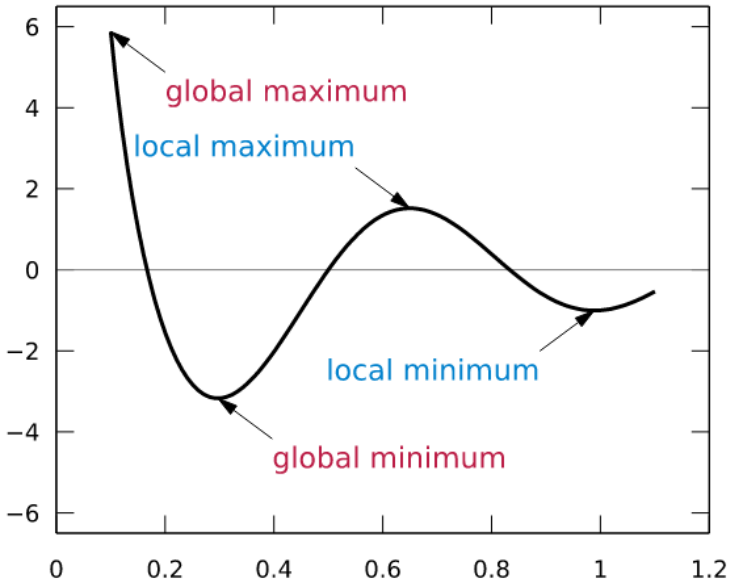


Figure (2.9) Stochastic Gradient Descent [7]

Appendix A shows in details an example of backpropagation of a neural network composed of one input layer, one output layer and two hidden layers.

2.2.4 Cost Function

A Feedforward Neural Network has multiple layers. It takes an input, that propagates through the network and then the neural network returns an output vector. More formally, it is called a_j^i the

activation (a.k.a. output) of the j^{th} neuron in the i^{th} layer, where a_j^1 is the j^{th} element in the input vector. Then we can relate the next layer's input to its previous via the following relation [10,11]:

$$a_j^i = \sigma\ (\ \Sigma_k\ (w_{jk}^i\ .\ a_k^{i-1}\ )\ +\ b_j^i\ ) \tag{2.6}$$

where σ is the activation function, $w_{jk}^i$is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer to the j^{th} neuron in the i^{th} layer. b_j^i is the bias of the j^{th} neuron in the i^{th} layer while a_j^i represents the activation value of the j^{th} neuron in the i^{th} layer.

Sometimes we write z_j^i to represent $\Sigma_k\ (w_{jk}^i\ .\ a_k^{i-1}\ )\ +\ b_j^i$, in other words, the activation value of a neuron before applying the activation function. Figure (2.10) shows a typical neural network that we are calculating the cost function for. For more concise notation we can write:

$$a^i = \sigma\ (w^i.\ a^{i-1}\ ) + b^i \tag{2.7}$$

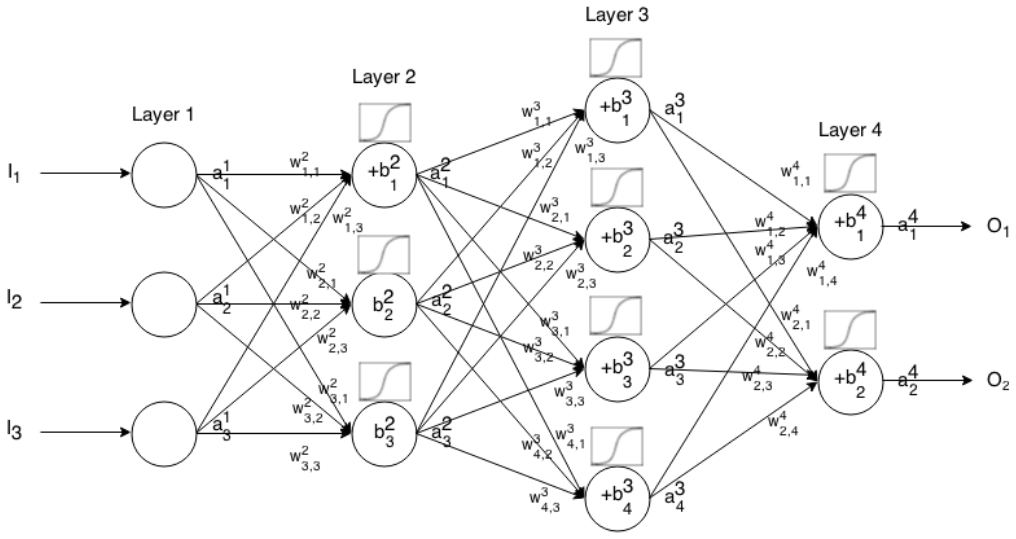


Figure (2.10) Cost Function in Neural Network [10]

To use this formula to compute the output of a feedforward network for some input I ∈ R_n, set a^1=*I*, then compute a^2, a^3, ... , a^m, where *m* is the number of layers. A cost function is a measure of "how good" a neural network did with respect to its given training sample and the expected output. It also may depend on variables such as weights and biases. It is a single value, not a vector, because it rates how good the neural network does as a whole. Specifically, a cost function is of the form:

$$C\,(W, B, S^r, E^r) \tag{2.8}$$

where *W* is our neural network's weights, *B* is our neural network's biases, S^r is the input of a single training sample, and E^r is the desired output of that training sample. Note this function can also potentially be dependent on y_j^i and z_j^i for any neuron *j* in layer *i*, because those values are dependent on *W*, *B*, and S^r. In backpropagation, the cost function is used to compute the error of our output layer, δ^L, via

$$\delta_j^L = \frac{\partial \mathrm{C}}{\partial a_j^L}\,\sigma'(z_j^i). \tag{2.9}$$

Which can also be written as a vector via

$$\delta^L = \nabla_a C \odot \sigma'(z^i) \tag{2.10}$$

In order to be used in backpropagation, a cost function can be defined as an average:

$$\mathrm{C} = \frac{1}{\mathrm{n}} \Sigma_x C_x \tag{2.11}$$

where C_x is the cost functions for each *x*, and *x* represents the individual training examples [10,11].

2.3 Machine Learning

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. It is basically a set of techniques that attempt to learn in multiple levels of representation, corresponding to different levels of abstraction that help to make sense of data such as images, sound, and text. It is typically implemented within artificial neural networks (ANN), as well as other classification and clustering algorithms. The levels in these learned statistical models correspond to distinct levels of concepts, where higher-level concepts are defined from lower-level ones, and the same lower-level concepts can help to define many higher-level concepts [14].

Machine learning has made significant contributions to society and has led to innovations in computer vision [5, 27], speech and image feature coding [5], phonetic recognition [5, 27], voice search [22], conversational speech recognition [22], semantic utterance classification [22, 27], natural language understanding [22], hand-writing recognition [28, 32, 34], audio processing, information retrieval [28], and robotics [14, 30].

The ability to independently adapt when exposed to new data is the main feature of machine learning. The iterative aspect of machine learning is mainly the reason why it has this ability. Recent advances in machine learning and signal and information processing research which have enabled the machine learning methods to effectively exploit complex, compositional nonlinear functions, to learn distributed and hierarchical feature representations [16,17].

2.4 Machine Learning and Neural Networks

There is a general confusion about the relationship between machine learning and neural networks. Neural network is a biologically-inspired programming paradigm which enables a computer to learn from observational data. Machine learning is a powerful set of learning techniques in neural networks [10]. Neural networks and machine learning currently provide the best solutions to many problems in image recognition, speech recognition, and social media.

Machine learning has three main types of algorithms: Supervised, Semi-supervised and unsupervised. The first one is used often when labeled data is provided in large amounts. One of the main advantages of this type its high detection rate, as it deals with new data based on certain given data. Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training (i.e. typically a small amount of labeled data with a large amount of unlabeled data). Unsupervised learning algorithms consist of mainly clustering (classification) algorithms as well as some special training methods of deep neural networks. [20,21].

2.5 Supervised Learning

Labeled data is tremendously useful. It enables training a model that is able to generalize and adapt to new data with high confidence and performance. Collecting information and labeling data is however difficult, expensive and labor intensive. Neural networks have a good track recorded in supervised learning and are commonly used [21].

2.5.1 Why Neural Networks?

Neural Networks derive their computing power through their massively parallel distributed structure and ability to learn. Furthermore, one of the most significant advantages of networks is their ability to generalize. It is actually their ability to produce reasonable outputs even for inputs that the network was not even trained on. Also, non-linearity is a positive property of neural networks as it enables it to adapt to non-linear functional relationships [9].

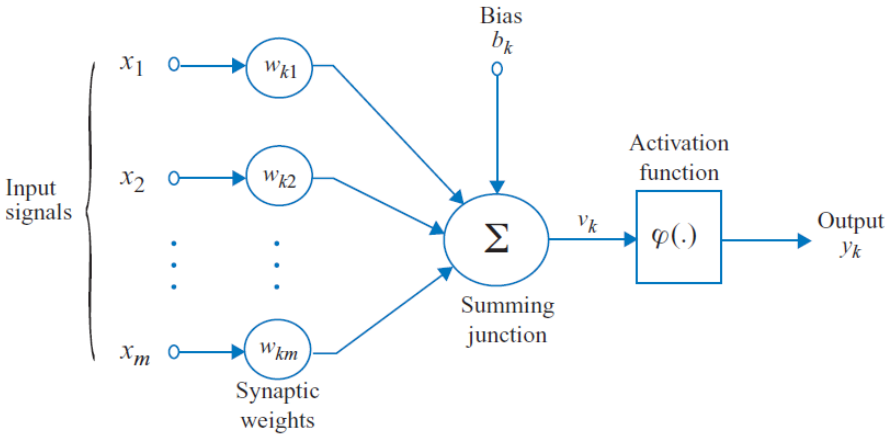


Figure (2.11) Non-linear model of a neuron [8]

The non-linear neuron of Figure (2.11) consists of an externally applied bias (b_k). It controls the increment or decrement of the net input of the activation function. This depends on whether it is positive (excitatory) or negative (inhibitory) respectively. The output value (v_k) produced at the end of the summing junction, including the bias, is referred to as the induced local field. A number of neurons are used and arranged into network structure. The network's input-output mapping is trained by using a set of labeled data enabling the network to mimic complex nonlinear functional relationships between input-output data. This is referred to as supervised learning (learning with a

teacher) that entails adapting the synaptic weights of a neural network. Every neuron in the network is potentially affected by the activity of other neurons [8]. Figure (2.12) shows the architecture of a neural network.

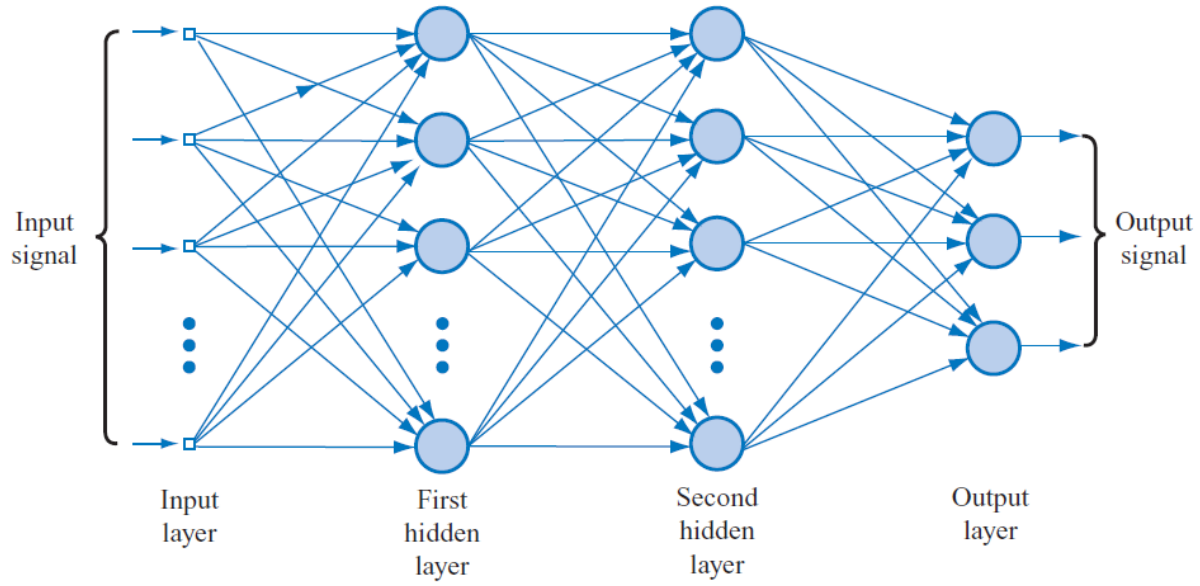


Figure (2.12) Neural Network Architecture [8]

Neural networks are composed of a number of interconnected nodes (i.e. neurons) organized in layers. All neural networks have one input layer, where the values of the input data are received. Processed results from each neuron in the input layer are sent to all the hidden neurons in the following layer (i.e. first hidden layer). Networks typically have at least one hidden layer, while the number of hidden layers differs from a network to another. The output from the output layer is thresholded to indicate the result.

2.5.2 Problems with Large Networks

In large neural networks with more than one hidden layer, the higher layers build new abstractions on top of previous layers. This means that larger networks can learn better in-practice rather than simple ones. But, increasing the number of hidden layers leads to two major problems. First, adding more hidden layers makes backpropagation less useful in passing information to the lower layers. Thus, the gradients begin to vanish and become relatively small in relation to the weights of the networks [6].

The second problem is overfitting, it is generally the central problem in machine learning. Overfitting describes the phenomenon of fitting the training data too closely to noise as well as information content of signals, maybe with hypotheses that are too complex [7]. The machine ends up fitting the training data really well including the noise, but the performance will be much poorer on real examples as it becomes very sensitive to minor fluctuations. G.E. Hinton et al. [19] showed that the overfitting problem occurs in large feedforward neural networks when it is trained on a small training set.

Deep learning is a powerful set of techniques for learning in deep neural networks, based on learning representations of data. It is also known as deep machine learning, hierarchical learning or deep structured learning. It is basically a set of algorithms modeling high-level abstractions in data using multiple processing layers with multiple linear and/or non-linear transformations. Stacked auto-encoders (AE), restricted Boltzmann machines (RBM) and deep belief networks (DBN) are examples of commonly used deep neural networks. They are basically feature detectors which attempt to overcome the problems of large networks.

The proposed solution was randomly omitting half of the feature detectors on each training case as it prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Thus, random "dropout" gives big improvements on many benchmark tasks and sets new records for speech and object recognition. MNIST is a large dataset of handwritten digits (see Figure 2.13) which stands for "Mixed National Institute of Standards and Technology" and is used to train learning algorithms. The machine learns how each digit looks like with different fonts, then it predicts what digit a new sample contains.

Figure (2.13) MNIST Handwritten digits [8]

Figure (2.14) represents the error rate on the popular MNIST dataset for a variety of neural network architectures trained with backpropagation using 50% dropout for all hidden layers. The upper set of lines is the best previously published result for this task using backpropagation while the lower set of lines use additional 20% dropout for the input layer as well. The improvement in the error rate is clearly significant. Figure (2.15) also shows the dramatic drop down in the frame classification error rate using the same changes in the hidden and input layers.

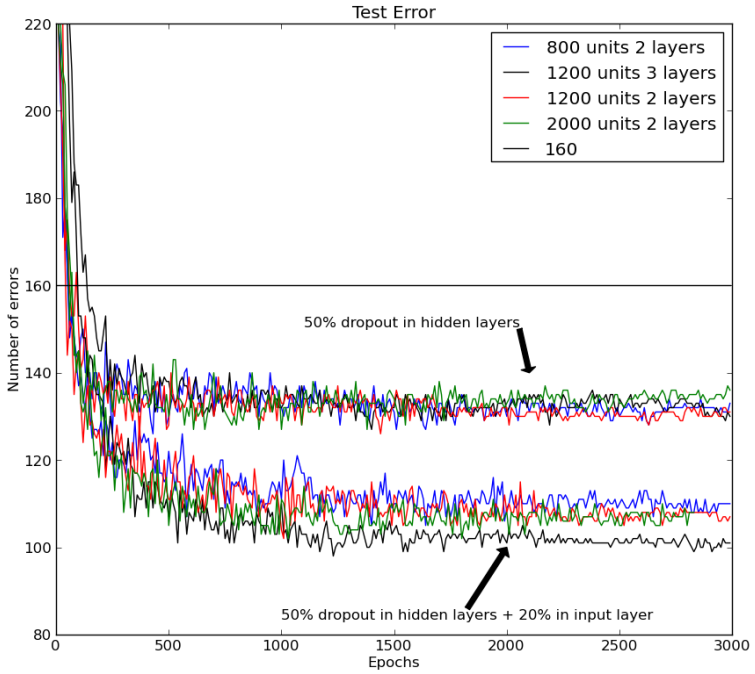


Figure (2.14) Changes effects on error rate [19]

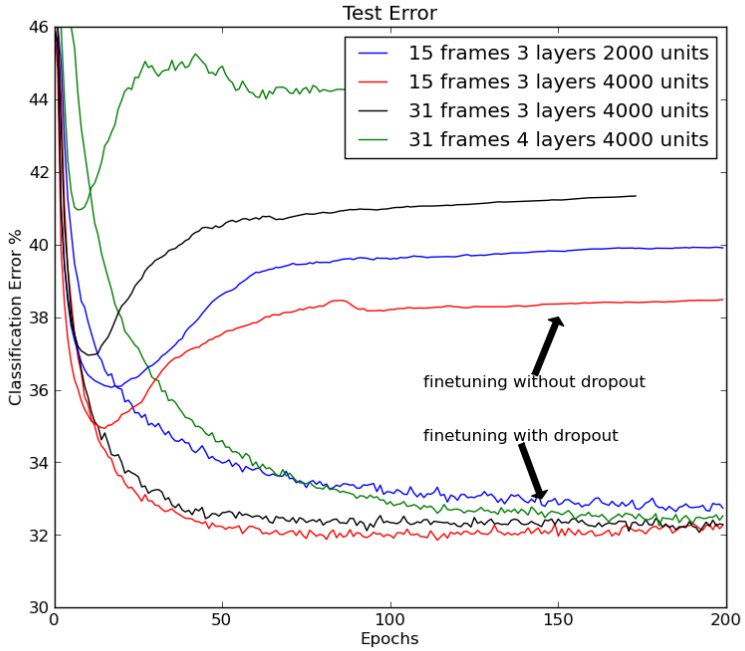


Figure (2.15) Changes effects on frame classification error rate [19]

2.5.3 Convolutional Networks

Convolutional neural networks (CNN) consist of convolutional and subsampling layers. The input to each layer is an image of dimension ***a* x *a* x *b***, where a is the height and width and b is the number of channels of the image (e.g. a rainbow image has b=7). Each layer has *k* learnable kernel filters of size ***m* x *m* x *n***, where $m < a$ and $n =< b$. A filter is basically a convolution matrix where each element of the image is multiplied with its local neighbors and weighted by the kernel. An input image is exposed to one or more image filters and the filter size gives rise to the whole structure that is convolved with the input image to produce *k* feature maps of size $\boldsymbol{a - m + 1}$. Each map is subsampled over *p* x *p* contiguous regions with mean (or max) pooling, where *p* starts from 2 for small input images and rises up to 5 for larger ones (See figure 2.16). Units of the same color have tied weights and units of different color represent different filter maps [81].

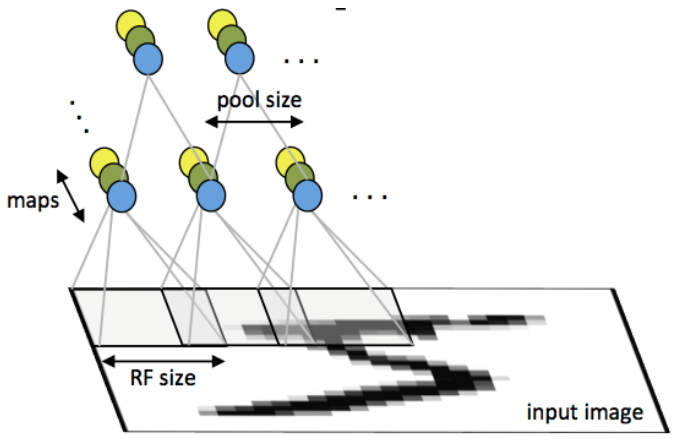


Fig (2.16) First layer of a convolutional neural network with pooling [81]

Figure (2.17) shows a particularly interesting and special class of feedforward networks used mostly in image recognition, the convolutional neural networks (CNN). Four 6x6 filters are built up to a given input image. The output pixel with coordinates 1,1 is the weighted sum of a 6x6 square of input pixels with top left corner 1,1 and the weights of the filter, which is also 6x6 square. Similarly, output pixel 2,1 is the result of input square with top left corner 2,1 and so on [7].

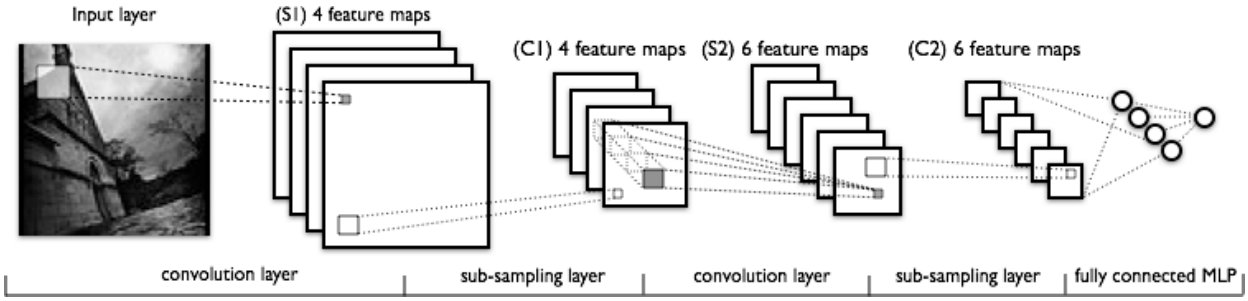


Figure (2.17) Convolutional Networks [7]

The following properties typically exist in convolutional network:

- **Convolutional layers** where a number of filters are applied to the input. As the figure shows, the first convolutional layer had four 6x6 filters, each filter results a feature map (FM). The filters are applied across all the feature maps of the previous convolutional layer with different weights in order to connect each input FM to each output FM. The shared weights help in detecting the features regardless of their location. However, the multiplicity of filters allows a different set of features to be detected by each filter.
- **Subsampling layers** which provide a compact representation of the input. The most popular ways of subsampling are max pooling, average pooling, and stochastic pooling [7].

- **The last convolutional or subsampling layer** is usually connected to one or more fully connected layers, where the last of which represents the labels.
- Modified backpropagation is used for training, taking into account subsampling layers. It also updates the shared weights based on all values to which that filter is applied [6].

2.5.4 Support Vector Machines

Support vector machines (SVM), a.k.a. support vector networks, are supervised learning models that can learn from data and identify objects. SVM is basically a binary linear classifier used for both classification and regression analysis [30]. SVM represents labeled training examples as points in space, mapped and divided by a clear gap that is as wide as possible. New samples are then mapped into the same space and assigned to the category they belong to, based on their location from the gap. SVMs can efficiently perform a non-linear classification, in addition to performing linear classification, by implicitly mapping their inputs into high-dimensional feature spaces, a.k.a kernel trick [30,31].

A decision plane is one that separates samples of a dataset having different class memberships. A schematic example is shown in the figure (2.18). The samples are painted in green and red and separated by a separating line made by a linear classifier. All points that lie on the right of this line are green, and those who are on the left are red. Same for new samples joining the dataset, will be classified to the same class of the side they lie in.

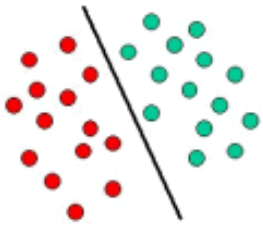


Figure (2.18) Decision plane

However, most classification problems are not that simple, and often more complex structures are needed in order to make an optimal separation that classifies new test samples on the basis of the examples of training samples. Compared to the previous schematic, it is clear in figure (2.19) that a full separation would be a curve not just a line. Support Vector Machines are typically suited to handle such tasks called hyperplane classifiers [30,31].

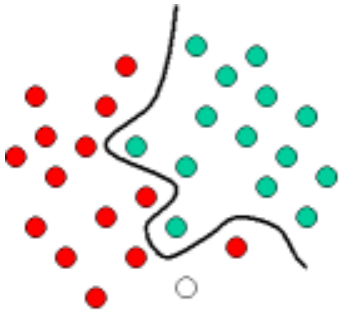


Figure (2.19) Curvy Separator

Figure (2.20) shows the basic idea behind Support Vector Machines. The original objects in the input space are rearranged using a set of mathematical functions, known as kernels. This transformation process from the input to the feature space is known as mapping. Note that the objects are linearly separable in the feature space. Thus, instead of constructing the complex curve, all we have to do is to find an optimal line that can separate green and red objects [30].

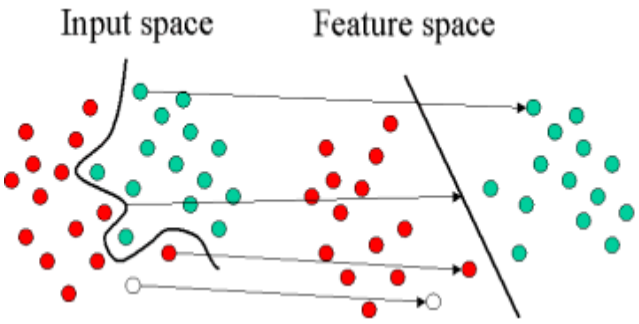


Figure (2.20) Support Vector Machines

Both regression and classification tasks are supported by SVM and multiple continuous and categorical variables can be handled. An iterative training algorithm is typically employed by SVM to construct an optimal hyperplane. This algorithm is mainly used to minimize an error function. Figure (2.21) shows the architecture of a SVM, where m_1 is the size of the hidden layer. Despite the way of implementation of a SVM, its conventional approach differs from the design of a multilayer perceptron in a fundamental way [3].

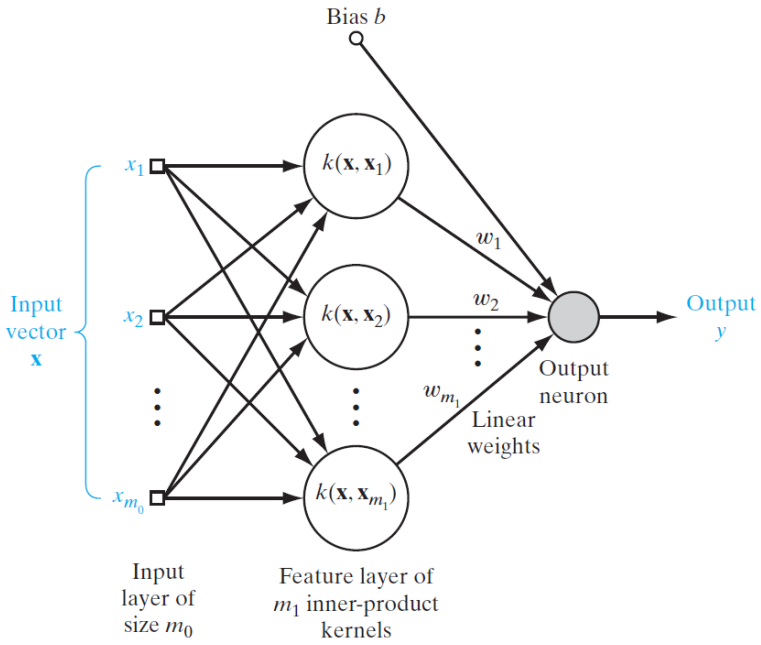


Figure (2.21) SVM architecture [8]

2.5.4.1 Selecting the Best Hyperplane

Assuming a set of data that contains a number of points of two different classes, the main objective of using a SVM is to identify the right hyperplane that best segregates the two classes. Since a hyperplane is just a line, it is possible to build up multiple hyperplanes that can perform this job as in figure (2.22).

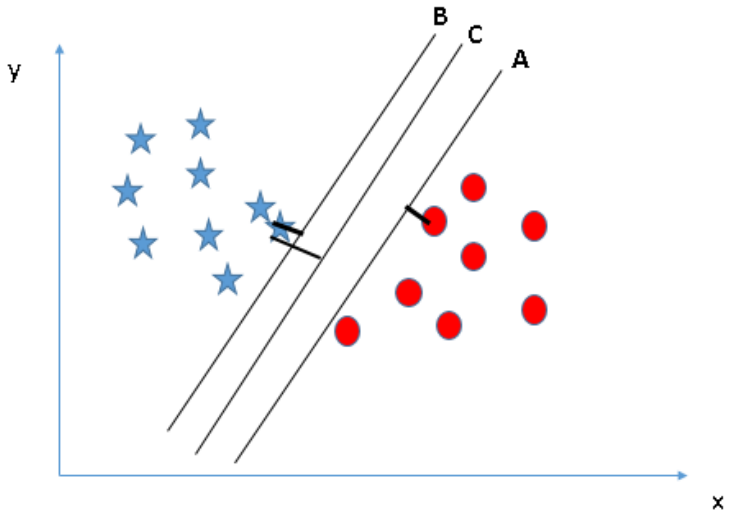


Fig (2.22) Multiple possible hyperplanes

The figure shows three lines (A,B and C) which all separate the red circles (class 1) from the blue stars (class 2) perfectly, but there must be a best separator. Since the represented data points are training points and it is expected to receive more samples during the testing process, thus the maximization of the margin (i.e. the distance between the nearest data point and the hyperplane) gives a better segregator. Obviously, hyperplane C has the highest margin as compared to A and B. On the other hand, selecting the highest margin must not have the priority rather than the separation of classes. In other words, SVM selects a hyperplane having a low margin and an accurate classification of all data points rather than a higher margin with misclassification.

However, some special cases would break the rule such as the problem in figure (2.23). A blue star lies in the territory of other class as an outlier. SVM in such case has a feature to ignore outliers and give the priority to hyperplanes with the maximum margin. This is what is called robustness of SVM, which is one of its advantages.

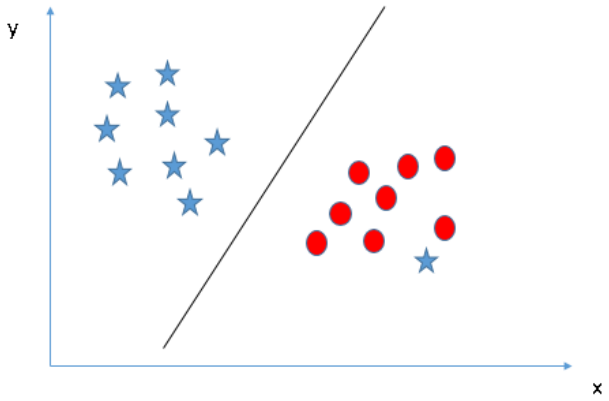


Figure (2.23) Outlier Case in SVM

The previous hyperplanes have been all linear so far. Figure (2.24a) gives an example of a more complicated problem, where we cannot separate data points with a straight line. SVM deals with this case by adding a new feature $z = x^2 + y^2$. Data points are replotted then on x axis and z axis in figure (2.24b).

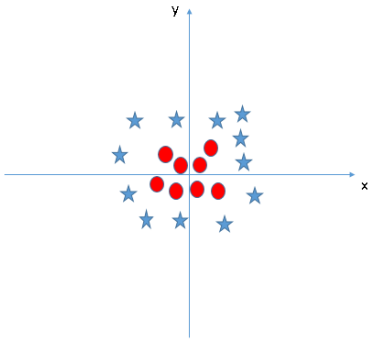


Figure (2.24a) Complicated problem in SVM

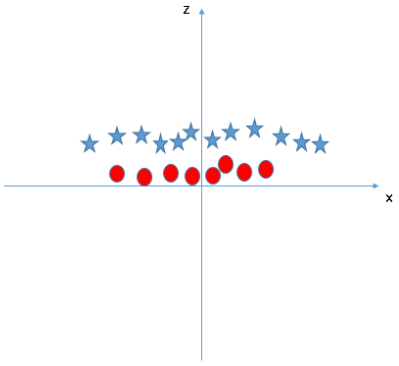


Figure (2.24b) Solution of Complicated Prob in SVM

Obviously, all values on z axis are positive since z is the squared sum of x and y. Red circles which were close to the origin on x and y lead to low values on z, while blue stars which were far away from the origin give higher values on z. At this stage, SVM sets a linear hyperplane and perform the regular job. This is the kernel trick that was mentioned in section (2.5.4), which takes low dimensional input space and transform it to a higher dimensional space. In other words, it converts inseparable problems to separable ones and it is used most in non-linear separation problems.

2.5.4.2 Multi-Class SVM

The most common technique to implement multi-class SVMs in practice is building |**C**| one-versus-rest classifiers (a.k.a. one-versus-all or OVA classification), where the sample is assigned to the class which classifies the test datum with the greatest margin. Another strategy is one-versus-one classifier, where the sample is compared with every classifier and assigned to the class the most ranked. This requires building $\frac{|\mathbf{C}|(|\mathbf{C}|-1)}{2}$ classifiers, which might decrease the training time since the training dataset for each classifier is much smaller.

A better approach that was used in this project is provided by the construction of multiclass SVMs, where a two-class classifier is built over a feature vector $\Phi(\vec{x}, y)$ where *x* is the sample vector and *y* is the target. The feature vector is derived from the pair consisting of the input features and the class of the datum. The classifier chooses the class:

$$y = \arg\max_{y'} \vec{w}^{\mathrm{T}} \Phi(\vec{x}, y') \tag{2.12}$$

where *w* is the weight vector and arg max are the arguments of the maxima (i.e. the points of the domain of following function at which the function values are maximized). The margin is the

distance between this value for the correct class and for the nearest other class. This method can be used for several kinds of linear classifiers. $\vec{w}^T\Phi(\vec{x},y')$ returns a signed real-valued value which can be interpreted as the distance from the separation (hyper) plane to the sample. This value can also be interpreted as a "confidence value". The larger the value the more confident one has that the sample belongs to the positive class. Thus, the sample is assigned to the class having the largest confidence value.

2.5.4.3 Advantages and Disadvantages of SVM

Table (2.1) shows the advantages and disadvantages of SVM which helps figuring out if it is the best algorithm for different applications:

Advantages	Disadvantages
• SVM works best with clear margin of separation. • Effective in high dimensional spaces. • Effective when the number of dimensions is greater than the number of samples. • Memory efficient as it uses a subset of training points (support vectors) in the decision function.	• Requires more training time with large datasets. • Does not perform very well with much noise (i.e. target classes are overlapping). • Probability estimates are not provided directly, they are calculated using an expensive five-fold cross-validation.

Table (2.1) Advantages and disadvantages of SVM

2.6 Semi-Supervised Learning

Semi-supervised learning is a learning paradigm concerned with the study of how computers and natural systems such as humans learn in the presence of both labeled and unlabeled data. It is basically considered as a special case of supervised learning. Combining labeled and unlabeled data may

change the learning behavior, and takes advantage of such a combination. It has potential as a quantitative tool for understanding human learning, where most of the input is self-evidently unlabeled. The success of semi-supervised learning depends critically on some underlying assumptions. At first glance, it might seem paradoxical that one can learn anything about a predictor $f: X \mapsto Y$ from unlabeled data. After all, f is about the mapping from sample x to label y, yet unlabeled data does not provide any examples of such a mapping. The answer lies in the assumptions one makes about the link between the distribution of unlabeled data $P(x)$ and target labels (i.e. labels of data). Figure (2.25) shows a simple example of how semi-supervised learning is possible [32,33].

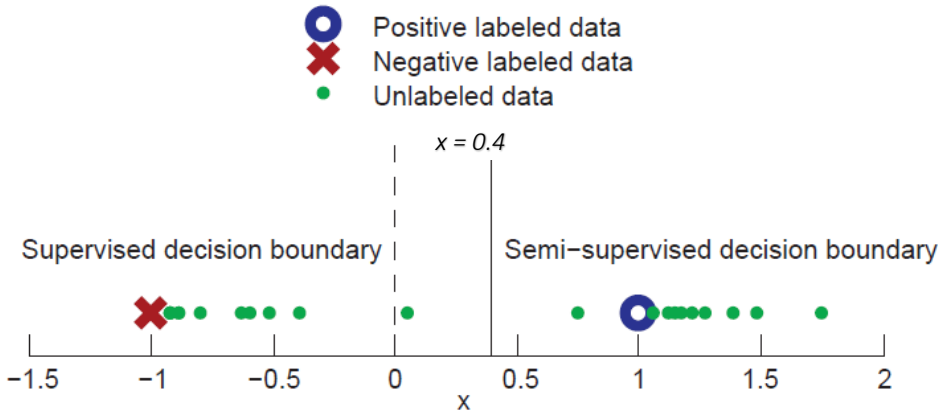


Figure (2.25) Semi-supervised learning [32]

where x is a one-dimensional value (or sample feature) $\in R$.

Assume a large number of unlabeled instances (i.e. green dots), where each sample is represented by a one-dimensional feature x. The samples are required to be classified into two classes: positive and negative. One positive labeled sample is marked with a blue hollowed circle while a negative one is marked with a red cross. In supervised learning, two labeled training instances x_1 and x_2 are provided where $(x_1, y_1) = (-1, -)$ and $(x_2, y_2) = (1, +)$. The best estimate of the decision boundary is obviously

$x = 0$ so that all instances with $x < 0$ should be classified as y = −, while those with $x \geq 0$ as y = +. It is observed that they form two groups. The assumption here was that the samples of each class form a coherent group, where $p(x|y)$ is a Gaussian distribution. The unlabeled data feeds the system with more information. The semi-supervised estimates the decision boundary to be between the two groups instead, at $x \approx 0.4$. If the assumption is true, then using both labeled and unlabeled data gives a more reliable estimate of the decision boundary. However, the distribution of unlabeled data helps to identify regions with the same label, and the few labeled data then provide the actual labels [32,33]. There are two main types of semi-supervised learning: Inductive and Transductive semi-supervised learning. The goal of the former is to predict the labels for future test data while the goal of the latter is to predict the labels for unlabeled samples in the training sample.

2.6.1 Inductive semi-supervised learning

For a training sample $\{(x_i, y_i)\}_{i=1}^{l}$, $\{x_j\}_{j=l+1}^{l+u}$, inductive semi-supervised learning trains a function $f : X \longmapsto Y$ such that f is a predictor for future data, beyond $\{x_j\}_{j=l+1}^{l+u}$. The performance on future data can be determined by using a separate data sample $\{(x_k, y_k)\}_{k=1}^{m}$ that is not actually used for training.

2.6.2 Transductive semi-supervised learning

In Transdusive learning, given a training sample $\{(x_i, y_i)\}_{i=1}^{l}$, $\{x_j\}_{j=l+1}^{l+u}$, a function $f : X^{l+u} \rightarrow Y^{l+u}$ is trained so that f is expected to be a good predictor of the unlabeled data $\{x_j\}_{j=l+1}^{l+u}$. f is defined only on the given training sample, thus is not required to make predictions outside.

Inductive semi-supervised learning is like an in-class exam, where the questions are not known in advance, and a student needs to prepare for all possible questions while transductive learning is like a take-home exam, where the student knows the exam questions and needs not prepare beyond those [32].

As a subject with many potential applications, semi-supervised learning uses a wide range of learning algorithms. One of the most common algorithms is the kernel approach based on **manifold regularization**. It is meant by the word "manifold," a *k*-dimensional topological space embedded in an *n*-dimensional Euclidean space where *n* is greater than *k.* If the functions describing the manifold are partially differentiable, then the manifold is called a "differentiable manifold". We may thus view the concept of a manifold as the generalization of the concept of a surface in and, by the same token, view the concept of a differentiable manifold as the generalization of a differentiable surface. The rationale for focusing on the kernel approach based on manifold regularization is threefold [3]:

1. The kernel approach for semi-supervised learning fits naturally within the scope of this chapter on regularization theory.

2. Manifold regularization provides a principled approach for the formulation of a data-dependent, nonparametric kernel for semi-supervised learning.

3. The use of manifold regularization has provided encouraging results on some classification tasks. Simply put, kernel-based manifold regularization has the potential to make a significant difference in semi-supervised learning theory.

The challenge in semi-supervised learning, discussed in a subsequent chapter, is to design a learning system that scales reasonably well for its implementation to be practically feasible when dealing with large-scale pattern classification problems. Reinforcement learning lies between supervised learning and unsupervised learning indeed, but it is totally different to semi-supervised learning. It is classified under unsupervised learning though, while the semi-supervised is identified as a kind of supervised learning. Reinforcement learning operates through continuing interactions between a learning system (agent) and the environment. The learning system performs an action and learns from the response of the environment to that action. In effect, the role of the teacher in supervised learning is replaced by a critic, for example, that is integrated into the learning machinery [3].

Results from a large-scale semi-supervised feature learning competition have been presented by D. Sculley [34]. The mission was the classification of malicious uniform resource locators URL (i.e. spam websites or viruses) with a sparse feature space of one million features, and training data sets of 50,000 labeled examples and one million unlabeled examples. Deep learning algorithms were used including Auto-encoders, RBMs and even decision trees.

2.6.3 Label Propagation Algorithm

Graph-based learning models are well-known with their fewest computation time. Label propagation is a graphical modeling algorithm for finding communities [76]. One of its best advantages is that it

doesn`t need priori information about the parameters of the network structure, as well as its fast computation. The main idea behind the label propagation algorithm is, suppose that a node *m* has a group of neighbor nodes: $m_1, m_2, \ldots\ldots m_k$. Each neighbor has the label of the community which it belongs to. Similarly, *m* takes on the label of its nearest neighbors. But on the other hand, *m* might have multiple communities around, which only one of them should be selected to take its label on. In label propagation method, the community which has the maximum number of neighbors around *m*, is the community that it will belong to. Connections with the minor communities get uniformly and randomly broken. Every node is first initialized with a unique label, then the labels are free to propagate through the network. During the label propagation process, the densest connected groups are the first to reach a consensus on a unique label. Figure (2.26) shows how labels expand from the originally labeled nodes to their neighbors.

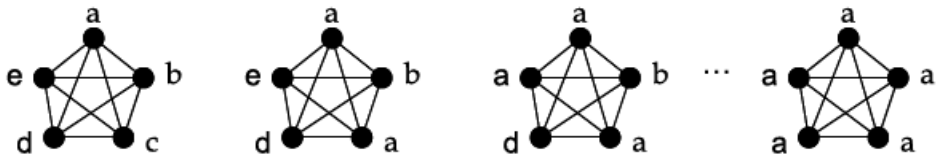


Figure (2.26) Label propagation among densely connected groups [76]

When more dense groups are formed throughout the network, they continue to expand outwards until it is possible to do so. Eventually, the propagation process ends up with all nodes having same labels are grouped together as one community. Performing this process one time ends with labelling all nodes, but this would not give the best performance. Most likely, the propagation process needs to be repeated in order to gamble the network again and achieve the optimum solution. Thus, this process is performed iteratively, where at each step, all nodes update their labels depending on their

neighbors. Synchronously, the nodes update their labels at the t^{th} iteration, based on their neighbors at the previous iteration (t-1).

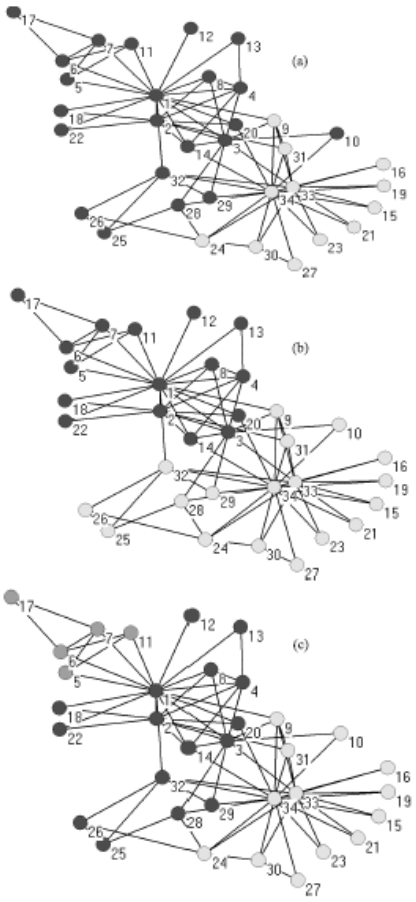


Figure (2.27) Different solutions for Zachary's karate club network [76]

The stop criterion is only a condition and not a measure that is being maximized or minimized. Consequently, label propagation algorithm does not give a unique solution. Figure (2.27) shows different solutions for Zachary's karate club network, which is a network of friendship among 34 members of a karate club. Over a period of time the club split into two factions due to leadership issues and each member joined one of the two factions. [76]

Nodes that are newly labeled by label propagation can have at least as many neighbors within their community as they have with each of the other communities, while those that are normally labeled (actual labels) in densely connected groups are attracted only to neighbors within their community. This is because it is assumed that actual labels from D&V Electronics are 100% trust worthy, while potentially new labels are predicted. Practically, the proposed label propagation algorithm can be implemented in the following steps [76]:

1. Labels at all nodes in the network should be initialized. For a given node *x*, $C_x(0) = x$. Where C_x is the label at the node *x.*

2. Set t = 1.

3. Arrange the nodes in the network in a random order and set it to *X*, where *X* is the total number of nodes.

4. For each $x \in X$ chosen in that specific order,

$$\text{let } C_x(t) = f\left(C_{xil}(t), \ldots, C_{xim}(t), C_{xi(m+1)}(t-1), \ldots, C_{xik}(t-1)\right) \tag{2.13}$$

Here, *f* returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly, *i* is the number of iterations and *l*, *m* and *k* are the neighbors.

5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set t = t + 1 and repeat (3).

When we start the label propagation algorithm with unique labels at each node, it should be noted that in the first few iterations, the network is divided into a number of small communities. As more iterations occur, the small communities become larger and larger by attracting more nodes to strengthen the community. The highest competitions run at the borders of each community, where the inner and outer groups tend to gain more members [77].

2.7 Unsupervised Learning

Learning without a teacher is not an easy task. Typically neural networks are not used for unsupervised learning. However, some special types can deal with unlabeled data. Q. V. Le et al. [36] explained in their Google's famous "cat" paper how they used special kind of deep autoencoders to distinguish human and cat faces by using unlabeled data. Javier Sanz, Ricardo Perera and Consuela Hueberta [38] proposed a method that combines the capability of wavelet transform (WT) to treat transient signals with the ability of auto-associative neural networks to extract features of data sets in an unsupervised mode. However, the unsupervised learning process can be implemented using two different types of algorithms: Classification or Clustering [37].

2.7.1 Classification Algorithms

Self-organizing learning is typically achieved in classification algorithms using Neural Networks. This section represents special types of these networks that can learn in an unsupervised manner.

2.7.1.1 AUTOENCODERS

Autoencoders are typically feedforward neural networks. The significance of this type comes from its tendency to reconstruct a compact representation (encoding) of a dataset.

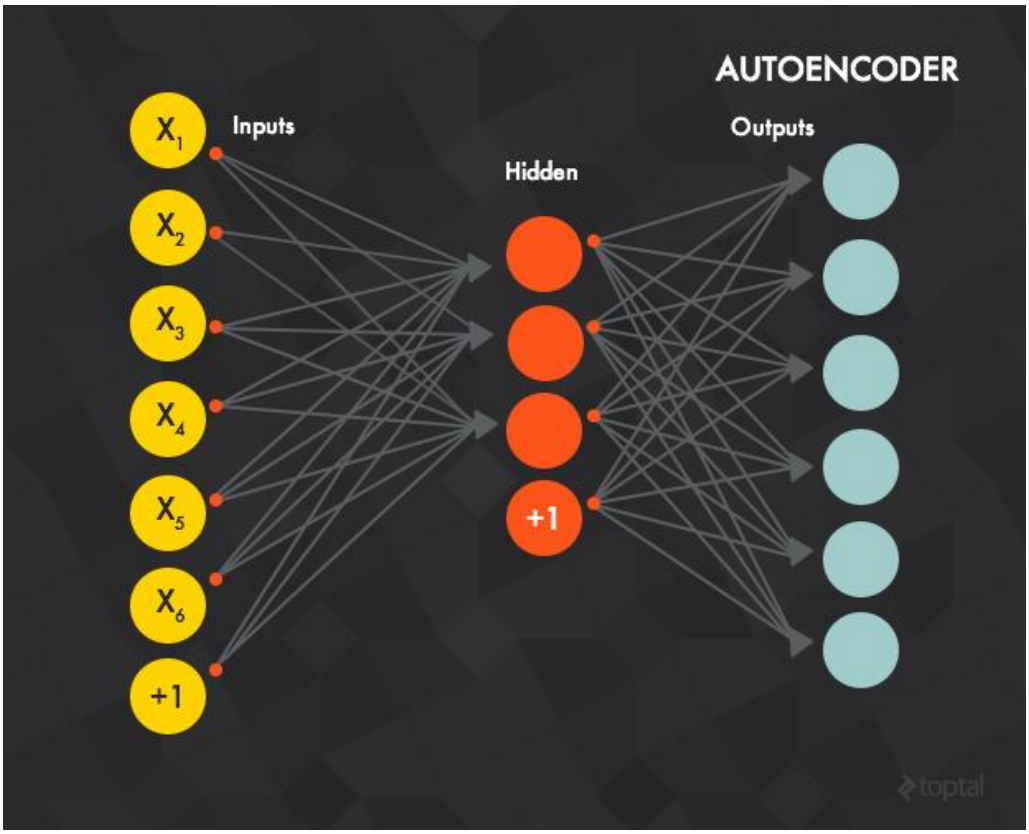


Figure (2.28) Autoencoder [7]

Figure (2.28) shows an autoencoder having 3 layers (input, hidden and output). It can be observed that the number of neurons in the hidden layer is less than those that are in the input layer. This looks as if some of the information of the input data are neglected somehow. That is because an autoencoder focuses on the features that best describe each sample. Thus, autoencoders tend to output the same input thing that entered the input layer of the network in a compressed way [7].

For example, if we are building a network to recognize whether the input data is an image of a car or a house. This is what we call "**two-class**" or "**binomial**" classification. The system will struggle to find entire parts in a hierarchical manner starting from tiny patterns of the image up to body parts ending by the entire image. Autoencoders in this case will find out wheels, front and rear doors, hood

and trunk to admit that this is a car regardless of its form (sedan or van, mini or jeep and regular or super car). The same scenario will be followed while recognizing a house. The system will look for entrance door, window and attic regardless of the color, decoration and size. The interests of an autoencoder varies depending on the goal of building the network. So if we are seeking to recognize the size, type, color and even the model of a car as shown in figure (2.29). Of course this is more difficult as the system distinguishes between the same things with minor differences. That is called "**multi-class**" classification [22].

Unsupervised learning is not only used for **classification**, but also for other tasks like **regression** and **anomaly detection**. Regression is when a system is trained on a number of values that are already known and what is required is to predict a value for the future (e.g. stock prices). However, sometimes the goal is to identify data points that are simply unusual. Anomaly detection is simply to learn what normal activity looks like using a history of similar actions and identify anything that is significantly different [25].

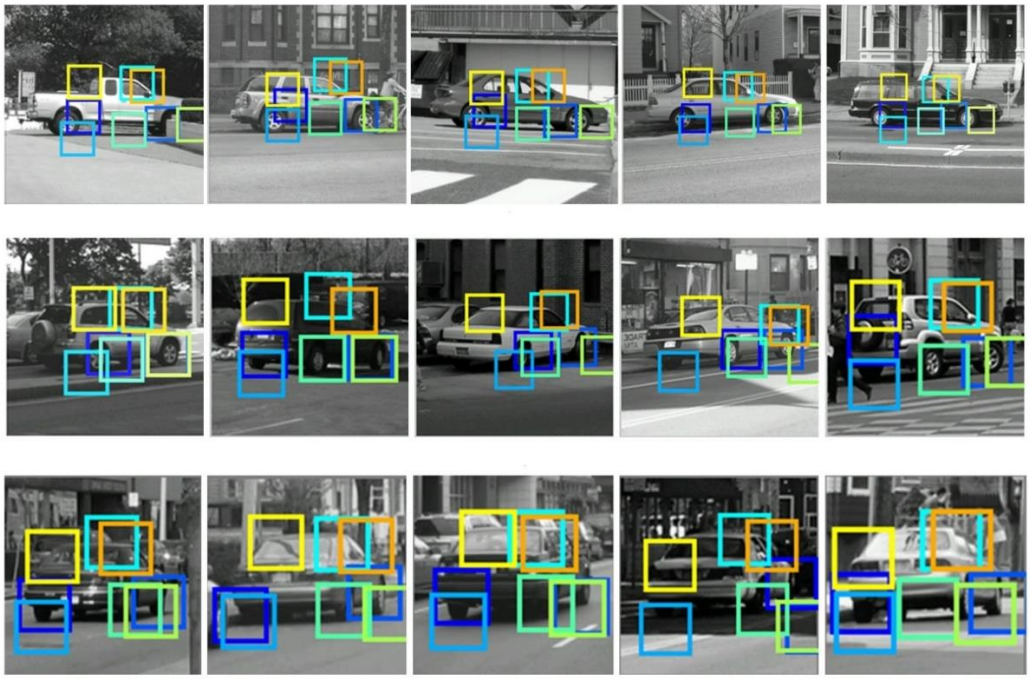


Figure (2.29) Multi-Class Classification

The intuition behind autoencoders is learning the internal structure and features of the data itself instead of a mapping between the training data and its labels. That is why the hidden layer in the middle is called "feature detector". The small number of hidden layers forces the network to learn only the most important features and achieves a dimensionality reduction. In other words, we are building a network with few small nodes that can produce a compact representation of the input data [6]. In fact, it is rare that we can use the features detected by autoencoders directly.

G.E. Hinton et al. [26] discovered that this structure can be stacked to form deep networks. It can be trained greedily, one layer at a time, to prevent vanishing gradient and overfitting problems associated with classic backpropagation as we are favoring a simpler representation rather than a

highly complex hypothesis that overfits the training data. The resulting structures are often quite powerful and give impressive results.

2.7.1.2 Stacked Autoencoders

Here is a simple model consisting of multiple stacked autoencoders shown in figure (2.30). Each entire autoencoder will be dealt with as a layer in a series of multiple autoencoders. The hidden layer of autoencoder *t* acts as an input layer to autoencoder *(t + 1)*. The input layer of the first autoencoder is the input layer for the whole network.

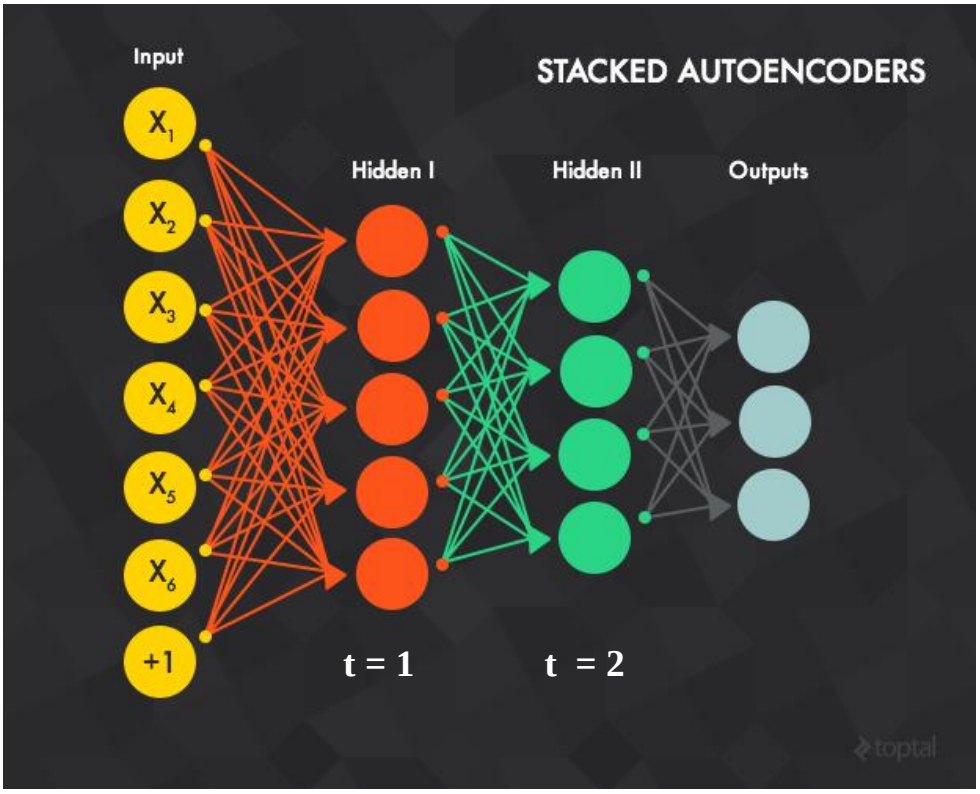


Figure (2.30) Stacked Autoencoders [7]

The training procedure starts by training the first autoencoder (*t=1)*, which is "Hidden I" with an additional output layer, individually using the backpropagation method. Then the second autoencoder (*t=2)* of "Hidden II". The output layer of each autoencoder is removed and replaced with yet another autoencoder. During this operation, the weights are updated using backpropagation and all the input training samples are used in each autoencoder. These steps are called *pre-training*. This procedure should be repeated then to all layers. However, there is no mapping between the input data and the output labels. It is important to note that it is still not possible to map the units from the last feature detector (i.e., the hidden layer of the last autoencoder) to get a practical output of the whole network. It is recommended to keep adding one or more fully connected layer until we get efficient results. The whole network can now be viewed as a multilayer perceptron and is trained using backpropagation (fine-tuning) [7]. Stacked auto encoders, then, are all about providing an effective pre-training method for initializing the weights of a network, leaving you with a complex, multi-layer perceptron that's ready to train (or *fine-tune*).

2.7.1.3 Restricted Boltzmann Machines

Restricted Boltzmann machines (RBM) are generative stochastic neural networks that can learn a probability distribution over their set of inputs. Figure (2.31) represents an RBM that can exist inside a neural network.

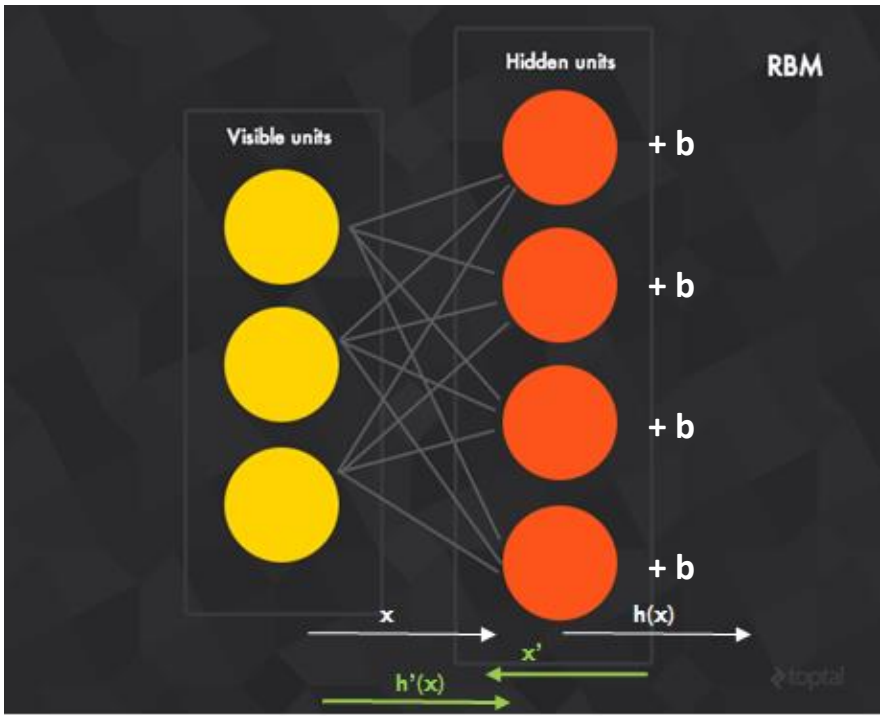


Figure (2.31) Restricted Boltzmann Machine [7]

New layer types used to show up in RBMs that are composed of a hidden layer, visible layer and bias. The connections between the visible and hidden layers are undirected, so that the values can be propagated in both direction. They are also fully connected, so that each unit in each layer is connected to each unit of the next layer only. In Boltzmann machines (BM), any unit in any layer is allowed to connect to those in any other layer. The unit activation of the hidden and visible layers of RBMs are binary (0 or 1) by default under a Bernoulli distribution. G.E. Hinton, [40] used other non-linearities to get variants for the unit activation of these layers. An unsupervised training algorithm is often used in RBMs instead of Backpropagation. It is called contrastive divergence (CD):

2.7.1.3.1 Contrastive Divergence

In this algorithm, two main phases are in place: Positive phase and Negative phase. The positive phase starts when an input sample (x) is clamped to the input layer. It is then propagated to the hidden layer exactly as in feedforward networks through an activation function h(x). As long as the connections between the visible and hidden layers are undirected (i.e. no arrows on connections), the second phase is when h(x) is propagated back to the visible layer to get (x'). Then, the new (x') is propagated back again to the hidden layer to get h(x)'. The weights of connections are updated as follows [7]:

$$\boldsymbol{w(t+1) = w(t) + a\,(x\,.\,h(x)^T - x'.\,h'(x)^T)} \tag{2.14}$$

where *a* is the learning rate, *t* is the layer number and *x, x', h(x), h'(x),* and *w* are vectors of the input sample, new sample, activation function, new activation function and weight respectively. The network has some perception of how the input data can be represented, so it tries to reproduce the data based on this perception. The main goal is for the generated data to be as close as possible to the real world and this is reflected in the weight update formula. That is achieved as the positive phase reflects the network's internal representation of the real world data while the negative phase represents an attempt to recreate the data based on this internal representation. If its reproduction is not close enough to reality, it makes an adjustment and tries again. After several hundred iterations, the same result as with autoencoders can be observed: one of the hidden units has a high activation value when, for example, "car" samples are presented, while the other is always more active for the "houses" samples [6].

2.7.1.4 Deep Belief Networks

Boltzmann machines can also be stacked as layers of a large network as with autoencoders. The difference is that contrastive divergence replaces the backpropagation method and RBMs replace autoencoders in deep belief networks (see Fig. 2.32) as well. The first RBM should be trained first. Following the same concept, its hidden layer acts as a visible layer for the next RBM and so on.

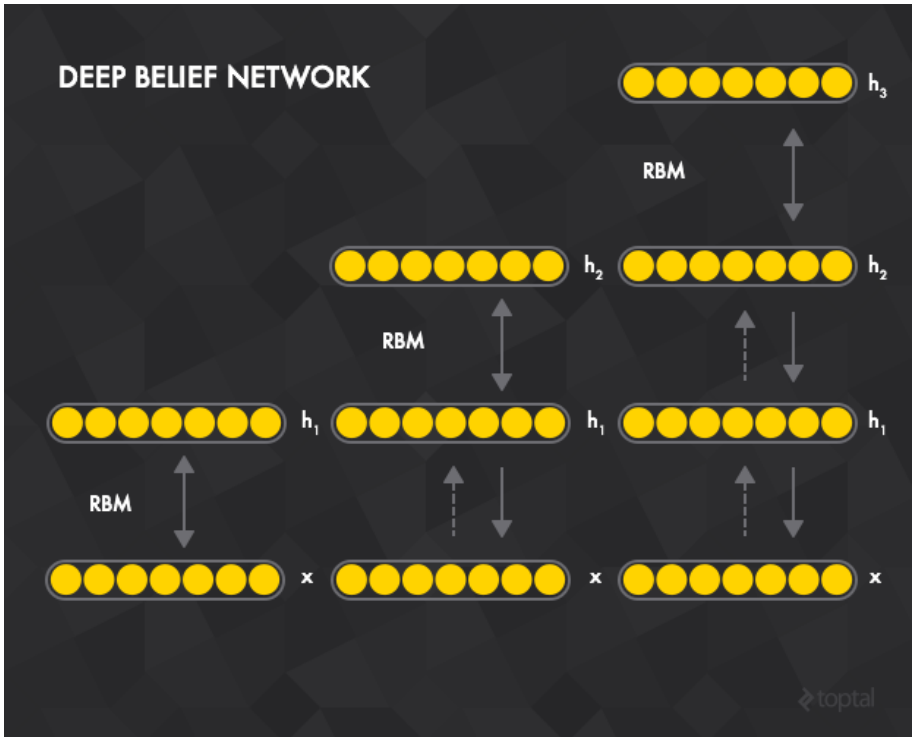


Figure (2.32) Deep Belief Networks [7]

After repetition of the pre-training process, the network can be extended by adding more fully connected layers to the hidden layer of the final RBM. Honglak Lee et al. proposed a combined scalable and generative unsupervised learning model of hierarchical representations with convolutional deep belief networks [42]. They showed that their model performed well in a variety of visual recognition tasks.

Neural networks in unsupervised learning are widely used to learn better representations of the input. This process does not come up with clusters of different features of the samples, but it creates meaningful representations that can be used for clustering. There is a number of different neural network architectures specifically designed for clustering such as self-organizing maps. However, other unsupervised clustering algorithms are proposed rather than neural networks [43].

2.8 Clustering Algorithms

Clustering is a process that classifies a given data set into homogeneous groups based on given features. It is the most important unsupervised learning problem as it gives the best results so far [36,42,43]. The main problem with data clustering algorithms is that they cannot be standardized. Clustering algorithms may give good result with one type of data set but may fail or give poor result with other types. There have been many attempts for standardizing these algorithms without much success. Many clustering algorithms have been proposed so far, and each algorithm has its own merits.

Some common conditions need to be satisfied for getting meaningful results from clustering algorithms. All the data must be scalable (i.e. in the same physical units). The clustering algorithm should be able to deal with different types of attributes and discover clusters with arbitrary shape. In addition, it should not be sensitive to noise and outliers. High dimensionality, interpretability and usability of the clustering algorithm are highly recommended as well [44,45].

Unsupervised Clustering algorithms can be broadly classified into two categories: Linear and Non-linear clustering algorithms. Linear clustering algorithms include K-means clustering, Fuzzy C-

means clustering, Hierarchical clustering, Gaussian (EM) clustering, Quality threshold clustering algorithms. Non-linear clustering algorithms include Minimum Spanning Tree (MST) based, Kernel K-Means, Density-based clustering algorithms. Another classification of clustering algorithms could be hard and soft clustering algorithms. In hard clustering, data is divided into distinct clusters, and each data element belongs to exactly one cluster. Data elements in soft clustering can belong to more than one cluster. Some of those algorithms will be explained in details as follows:

2.8.1 K-means Clustering Algorithm

The input training set $\{x^{(1)}, \ldots, x^{(m)}\}$ is expected to be unlabeled as a clustering problem. K-means algorithm is a linear hard clustering algorithm. The main objective of this problem is to group the data into cohesive clusters, where $x^{(i)} \in \mathrm{R}^n$ as usual. The algorithm occurs as follows [44]:

a. Initialize Cluster Centroids where, in this step, some cluster centroids $(\mu_1, \mu_2, \ldots, \mu_k) \in \mathrm{R}^n$ are assumed random, where k is the number of clusters that needs to be found.

b. Repeat until convergence:

{ For each i, set $$c^{(i)} = arg\,min_j \, ||x^{(i)} - \mu_j||^2 \quad (2.15)$$

For each j, set $$\mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\}\, x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}} \quad (2.16)$$

}

where μ_j is the cluster centroids that represent the current guesses for the positions of the centers of the clusters. In order to initialize the cluster centroids in the first step of the algorithm, k training

samples are randomly distributed. The cluster centroids should be equal to the values of these k samples. Two main steps are carried out repeatedly during running this algorithm [45]:

a. Cluster Assignment where random centroids are selected and each point $x^{(i)}$ is classified to the nearest cluster centroid μ_j.

b. Moving Centroid where the cluster centroid μ_j, moves to the average point of all the classified points of the corresponding cluster k. The two steps should be repeated until no further change happens. The following figure (2.33) shows an example of the different stages during K-means algorithm:

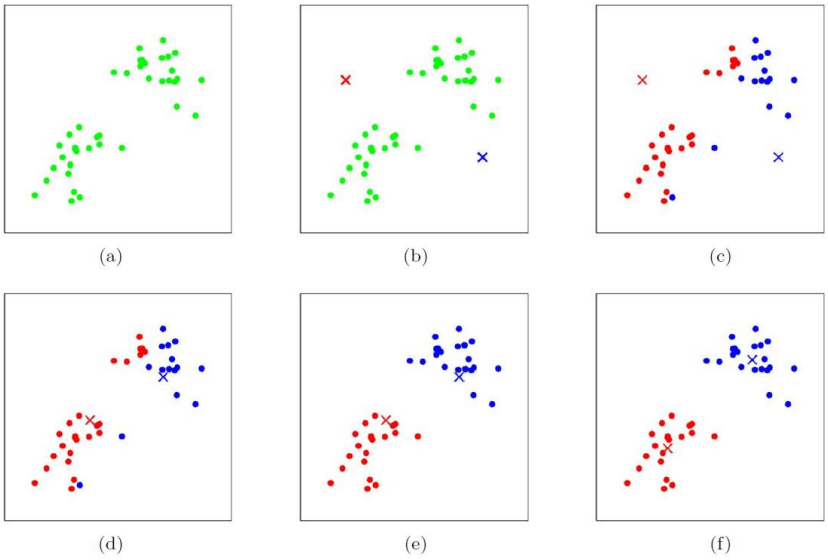


Figure (2.33) K-Means stages

The first picture (a) is the original training dataset plotted as green dots. The next one (b) is a random initialization of cluster centroids plotted as crosses. Note that the cluster centroids are not chosen to be equal to training samples. From (c) to (f), four iterations of K-means are running. Each training examples is assigned to the closest cluster centroid. It can be observed that two different colors are

used to distinguish the two different clusters. One of the clusters is painted in blue and the other one in red. After that, each centroid is moved to the mean of all the points that have been assigned to it [46,47].

Figure (2.34) shows an example of MNIST handwritten digits dataset, clustered by K-Means clustering algorithm using Scikit-learn library in Python programming language. The dimensionality of this dataset was pre-reduced using Principal Component Analysis that will be discussed in details later. The centroids are marked with white crosses.

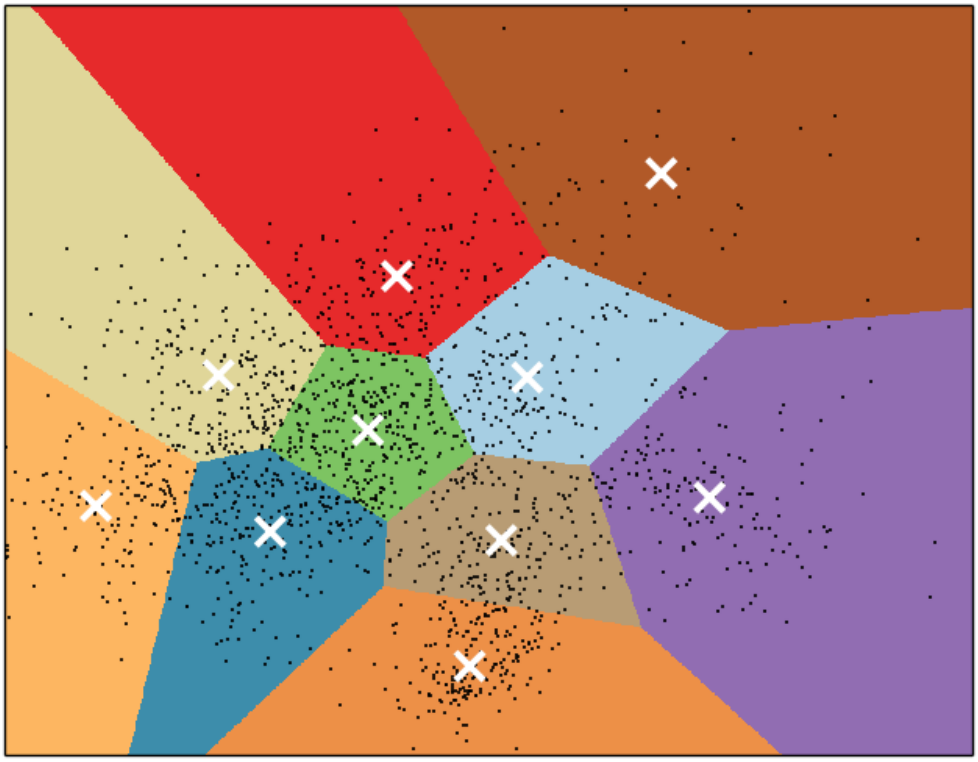


Figure (2.34) PCA-reduced data K-Means Clustering Algorithm [48]

Adam Coates and Andrew Y. Ng, [50] found that K-means clustering can be used as a fast training method. The main advantage of this approach is that it is very fast and easily implemented. K-Means will require significantly increased amounts of data, possibly negating its speed advantage.

2.8.2 Kernel K-Means Clustering Algorithm

Kernel K-Means in a non-linear clustering algorithm. It applies the same trick as k-means but the difference is that in the calculation of distance, kernel method is used instead of the Euclidean distance. This is where the non-linearity is showing up from. Let $X = \{a_1, a_2, a_3, ..., a_n\}$ be the set of data points and *c* be the number of clusters. First step is the random initialization of '*c'* cluster centers. Then, the distance of each data point and the cluster center in the transformed space should be computed using the following equation:

$$D(\{\pi_c\}_{c=1}^{k}) = \sum_{c=1}^{k} \sum_{a_i \in \pi_c} ||\Phi(a_i) - m_c||^2, \text{ where } m_c = \frac{\sum_{a_i \in \pi_c} \Phi(a_i)}{\pi_c}.$$

$$\Phi(a_i).\Phi(a_i) - \frac{2\sum_{a_i \in \pi_c} \Phi(a_i).\Phi(a_i)}{|\pi_c|} + \frac{2\sum_{a_j a_i \in \pi_c} \Phi(a_i).\Phi(a_i)}{|\pi_c|^2} \quad (2.17)$$

where, c^{th} cluster is denoted by π_c, *'m_c'* is the mean of the cluster π_c and *'$\Phi(a_i)$'* is the data point a_i in transformed space.

$$\Phi(ai).\Phi(aj) = \begin{cases} e^{||ai - aj||)*q}, & for\ gaussian\ kernel \\ (c + ai.aj)^d, & for\ polynomial\ kernel \end{cases} \quad (2.18)$$

After that, data point is to be assigned to the cluster center whose distance is minimum. As long as data points are not re-assigned, computation step should be repeated. This algorithm is able to identify

the non-linear structures and is best suited for real life dataset. In order to apply this technique, the number of cluster centers needs to be pre-defined. The main disadvantage of Kernel K-Means is its large complexity [44,45].

2.8.3 Fuzzy C-means (FCM) Clustering Algorithm

FCM (developed by Dumn, 1973 and deveopled by Bezdek, 1981) is a linear data clustering technique in which a dataset is grouped into n clusters with every data point in the dataset belonging to every cluster to a certain degree. This is what we called soft clustering. In other words, it allows the same data point to belong to one or more clusters and associated with each element is a set of membership levels. These indicate the strength of the association between that data element and a particular cluster. A certain data point that lies close to the center of a cluster will have a high degree of belonging or membership to that cluster and another data point that lies far away from the center of a cluster will have a low degree of belonging or membership to that cluster. This method is frequently used in pattern recognition [52,53].

Fuzzy clustering assigns these membership levels, and then using them to assign data elements to one or more clusters. It attempts to partition a finite collection of n elements $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ into a collection of C fuzzy clusters with respect to some given criterion. Given a finite set of data, FCM returns a list of cluster centres $C = \{\mathbf{c}_1, ..., \mathbf{c}_c\}$ and a partition matrix $W = w_{i,j} \in [0,1],\ i = 1, ..., n,\ j = 1, ..., c$, where each element w_{ij} tells the degree to which element $\mathbf{x}_i$ belongs to cluster $\mathbf{c}_j$. As in K-means clustering, the FCM aims to minimize an objective function:

$$\arg\min_{C} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \left\| \mathbf{x}_i - \mathbf{c}_j \right\|^2, \tag{2.19}$$

Where,

$$w_{ij}^{m} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}}.$$

The difference here is adding membership function values w_{ij} and the fuzzifier $m \in R$ with $m \geq 1$ The fuzzifier m determines the level of cluster fuzziness. A large m results in smaller memberships w_{ij} and hence, fuzzier clusters. In the limit $m = 1$, the memberships w_{ij} converge to 0 or 1, which implies a crisp partitioning. In the absence of experimentation or domain knowledge, m is commonly set to 2 [54]. Any point *x* has a set of coefficients giving the degree of being in the *k*th cluster $w_k(x)$. With fuzzy *c*-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster:

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m}. \tag{2.20}$$

The degree of belonging, $w_k(x)$, is related inversely to the distance from *x* to the cluster center as calculated on the previous pass. It also depends on a parameter *m* that controls how much weight is given to the closest center. The fuzzy *c*-means algorithm is very similar to *k*-means algorithm [57]:

It starts by choosing a number of clusters, then each point is assigned to its nearest cluster. This is repeated until convergence, according to a sensitivity threshold ε). Then, the centroid c_k of each

cluster is computed and for each point, its degree of membership in the clusters are computed. Figure (2.35) shows an example of a clustered dataset using FCM.

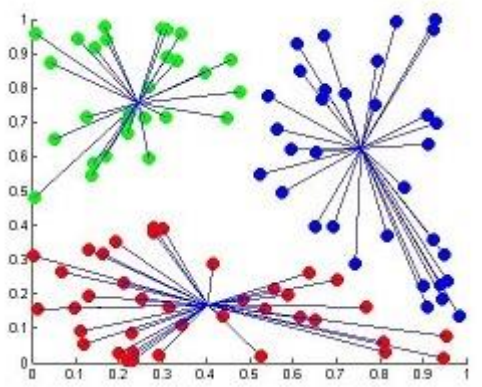


Figure (2.35) Fuzzy C-means Algorithm [58]

FCM minimizes intra-cluster variance as well, but has the same problems as *k*-means. The minimum is a local minimum, and the results depend on the initial choice of weights. Using a mixture of Gaussians along with the expectation-maximization algorithm is a more statistically formalized method which includes the idea of partial membership in classes. This algorithm has been a very important tool for image processing in clustering objects in an image [60].

2.8.4 Hierarchical Clustering Algorithm

Hierarchical linear clustering algorithm is mainly including two types: Agglomerative Hierarchical clustering algorithm (a.k.a. AGNES, i.e. agglomerative nesting) and Divisive Hierarchical clustering algorithm (a.k.a. DIANA, i.e. divisive analysis). Those two algorithms are exactly the reverse of each other [42].

Agglomerative Hierarchical clustering works by grouping the data one by one on the basis of the nearest distance measure of all the pairwise distances between data points. Many methods are available to calculate this distance such as single-nearest distance (single linkage), complete-farthest distance (complete linkage), average-average distance (average linkage), centroid distance and ward's method where the sum of the squared Euclidean distance. In this project, a centroid-based algorithm, known as Mean Shift algorithm, was used. It involves shifting the kernel of the densest area iteratively to a higher density region until convergence. Mean Shift algorithm is computationally intensive and takes a longer processing time than others techniques. It also relies on sufficient high density of data with a clear gradient to locate the cluster centers. It is considered to be more reliable than the K-Means algorithm and does not suffer from crisp clustering. In other words, it gives more freedom for samples to move between different clusters [62].

The data should be grouped until one cluster is formed. Based on dendogram graph, this algorithm calculates how many number of clusters should be created. The following steps occur during this algorithm:

Assumed, $X = \{x_1, x_2, x_3, ..., x_n\}$ as the set of data points, the first step is the disjoint clustering having level L= 0 and sequence number m = 0. Then the least distance pair of clusters in the current clustering should be found. For example, assume pair (r) and (s), according to d[(r),(s)] = min d[(i),(j)] where the minimum is over all pairs of clusters in the current clustering. Increment the sequence number: m = m +1. Merge points (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to L(m) = d[(r),(s)].

The distance matrix D should be then updated by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The distance between the new cluster (r,s) and old cluster (k) will be defined as d[(k), (r,s)] = min (d[(k),(r)], d[(k),(s)]). If all the data points are in one cluster the system will stop, if not, it should repeat again starting from finding the least distance pair of clusters step. Divisive Hierarchical clustering is typically the reverse of this approach [42,62].

Hierarchical clustering has some advantages: The number of clusters does not have to be pre-determined, easy to implement and gives best results in some cases. But on the other hand, some drawbacks show up as paybacks for these advantages. This algorithm unfortunately can never undo what was done previously. Time complexity of at least ($n^2 \log n$) is required, where n is the number of data points. Furthermore, based on the type of distance matrix chosen for merging, different algorithms can suffer with one or more of the following issues: i) sensitivity to noise and outliers, ii) breaking large clusters, iii) difficulty in handling different sized clusters and convex shapes, iv) no objective function is directly minimized, and sometimes, difficulty in identifying the correct number of clusters [62-64].

A map of some Italian cities [84] is shown in figure (2.36). The objective is to perform hierarchical clustering of distances (in kilometers) between these cities. The input distance matrix is shown below in table (2.2) at a level L=0 for all clusters and a sequence number m=0.

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

Fig (2.36) Hierarchical Clustering Example (a) **Table (2.2) Distance between Cities Example (a)**

Obviously, MI and TO are the nearest pair of cities, at a distance of 138 Km. They are merged into one cluster called "MI/TO" as in figure (2.37). The level of the new cluster changed to L(MI/TO) = 138 and the sequence number is now m=1. Then the distance from this cluster to all other cities is calculated (i.e. the shortest distance from any member of the cluster to every city) in table (2.3). For example, the distance from "MI/TO" to RM is actually from MI to RM (i.e. L(MI/TO)=564 Km), and so on.

	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0

Fig (2.37) Hierarchical Clustering Example (b) **Table (2.3) Distance between Cities Example (b)**

The process keeps running until the last two clusters are merged (see fig 2.38) at L(BA/FI/NA/RM/MI/TO)=295 (see table 2.4) and m=5.

	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0

Fig (2.38) Hierarchical Clustering Example (c) **Table (2.4) Distance between Cities Example (c)**

The entire process is summarized and described by the following hierarchical tree (a.k.a. dendogram) as shown in figure (2.39).

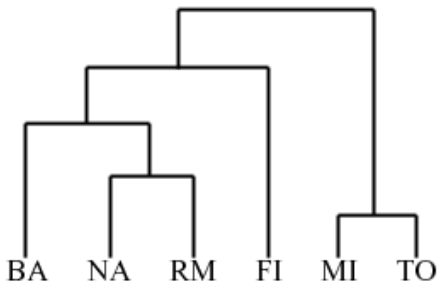


Fig (2.39) Hierarchical Clustering Dendogram Example [42]

2.8.5 Gaussian Expectation Maximization (EM) Clustering Algorithm

This algorithm assumes that there are *n* Gaussian and tries to fit the data into the *n* Gaussian by expecting the classes of all data point and then maximizing the maximum likelihood of Gaussian centers. Let $X = \{x_1, x_2, x_3, ..., x_n\}$ be the set of data points, $V = \{\mu_1, \mu_2, \mu_3, ..., \mu_c\}$ be the set of means of Gaussian and $P = \{p_1, p_2, p_3, \ldots, p_c\}$ be the set of probability of occurrence of each Gaussian. The following steps should be followed to implement Gaussian (EM) algorithm:

1. On the p^{th} iteration initialize (a.k.a. Expectation (E) step):

$$\lambda_t = \{\mu_1(t), \mu_2(t) \ldots \mu_c(t), \sum_1(t), \sum_2(t) \ldots \sum_c(t), \rho_1(t), \rho_2(t) \ldots \rho_c(t)\} \quad (2.21)$$

where μ_n is the mean value and *p* is a superscript that marks all parameters that have been estimated at the *p*th iteration.

2. Compute the expected classes of all data points for each class using:

$$P(w_i|x_k, \lambda_t) = \frac{p(x_k|w_i, \lambda_t)\, P(w_i|\lambda_t)}{p(x_k|\lambda_t)} = \frac{p(x_k|w_i, \mu_i(t), \sum_i(t))\, p_i(t)}{\sum_{j=1}^{c} p(x_k|w_j, \mu_i(t) \sum_i(t))\, p_i(t)} \quad (2.22)$$

Where *P* is the bayesian probability, *x* is the data point, *w* is the weight and λ is the class.

3. Compute "maximum likelihood" given our data class membership distribution using (a.k.a. Maximization (M) step):

$$\mu_i(t+1) = \frac{\sum_k P(w_i|x_k, \lambda_i) x_k}{\sum_k P(w_i|x_k, \lambda_i)} \tag{2.23}$$

$$p_i(t+1) = \frac{\sum_k P(w_i|x_k, \lambda_i)}{R} \tag{2.24}$$

where R is the number of data points and μ is the maximum likelihood.

4. Iterate steps 2 and 3 until convergence.

One main advantage for this algorithm is that it gives extremely useful results for the real world data set. However, it is highly complex to compute [68]. Table (2.5) illustrates a simple comparison between K-means, Hierarchical and Gaussian Mixture clustering algorithms. Figure (2.40) also exhibits graphically the difference between K-Means clustering as a hard clustering algorithm and agglomerative hierarchical clustering (i.e. introduced in section 2.8.4) as a soft clustering algorithm. This is the popular Swiss Roll dataset using Matlab programming language.

Learning Method	Loss Function	Number of clusters: Predetermined or Data-dependent	Cluster shape: isotropic or anisotropic?	Parameter Estimation Algorithm
K-means	Within-class squared distance from mean	Predetermined	Isotropic	K-means
Gaussian Mixture Models (identity covariance)	$-\log P(X)$, (equivalent to within-class squared distance from mean)	Predetermined	Isotropic	Expectation Maximization (EM)
Single-Link Hierarchical Clustering	Maximum distance between a point and its nearest neighbor within a cluster	Data-dependent	Anisotropic	Greedy agglomerative clustering

Table (2.5) Comparison between Clustering Algorithms [70]

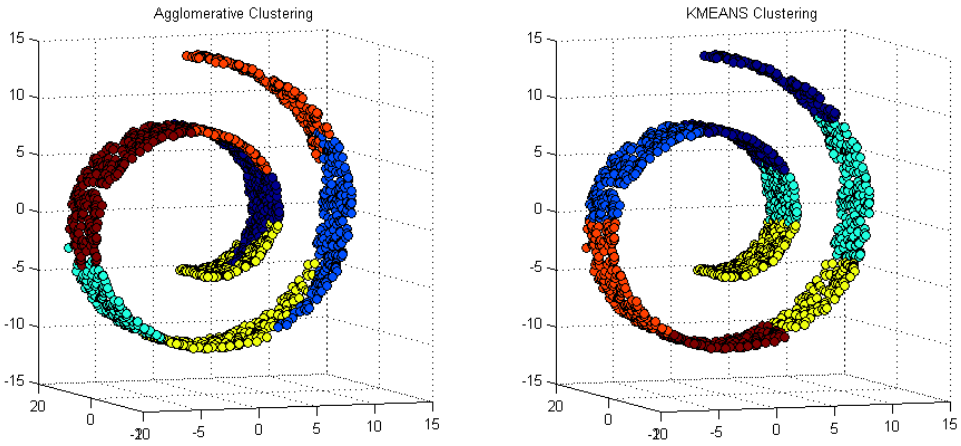


Figure (2.40) K-Means vs Agglomerative Methods [72]

2.9 Dimensionality Reduction

Clustering algorithms can be represented visually in 2 or 3 dimensional scatter plats while the training samples might be in higher dimensions. This case requires converting the multi-dimensional data into lower dimensions that can be visually observed and easily analyzed. Principle Components Analysis (PCA) is the most common algorithm used to reduce the dimensionality. This task is not only for a better visual representation but to reduce also processing time that sometimes needs high memory requirements.

2.9.1 Principle Components Analysis (PCA)

The concept behind PCA is clear from its name. It seeks to analyze the data in order to get a compact representation. Orthogonal transformation is used in this algorithm to convert the dependent set of measurements into a set of independent principle components. Let us assume that a set of dependent measurements defined in a matrix form as $X_k = (x_1, x_2 \ldots\ldots, x_p)_k$, having p variables with k measurements for each variable. Each column of the matrix represents k measurements for a single sensor, such as x_1. A transformation is used as follows [2]:

$$T = X P \tag{2.25}$$

where P is a Principle Components Loading matrix, which when multiplied by another matrix (i.e. X), it transforms it to its principles components scores. The output of this equation produces the principle components scores that are uncorrelated signals to the cross-correlated signals X. Each column in this matrix represents k measurements for one variable t_i. The first column has the highest variance component t_1, and it decreases gradually to the lowest variance component t_p. This means that the first few principle components carry the highest amount of information while the last few

ones are low variance variables that carry the least amount of information. Thus, PCA is giving the measurement signals that contains the most significant information. Measured data that would not be useful will be eliminated. The transformation process can be illustrated by mapping the measurements onto orthogonal axes [2].

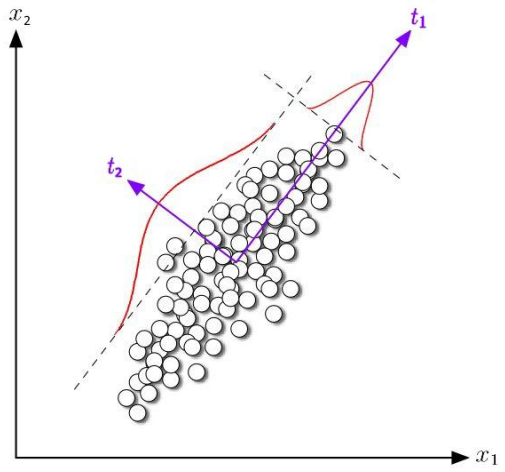


Figure (2.41) Principle Component Analysis [2]

Figure (2.41) shows the principle component analysis where x_1 and x_2 are the input values and t_1 and t_2 are the output principle components. As shown, the principle components are orthogonal, and t1 axis has higher variance than t2. The output of PCA brings out the strongest patterns that best represent the observed variables in a dataset. Basically, this output does not actually have any physical meaning, it simply consists of weighted combinations of the original variables. The mathematical bases should be orthogonal and result in uncorrelated outputs in order to get a transformation matrix *T*. The sample covariance of *T* is [74]:

$$\Sigma_T = \frac{1}{n-1} T^t T \tag{2.26}$$

$$= \frac{1}{n-1} (XP)^t XP \tag{2.27}$$

$$= \frac{1}{n-1} P^t (X^t X) P \tag{2.28}$$

Where the term ($X^t\ X$) results in a symmetric matrix *S:*

$$S = (X^t\ X) \tag{2.29}$$

This symmetric matrix can be then decomposed into its eigenvalues and eigenvectors:

$$S = (B \Lambda\ B^{-1}) \tag{2.30}$$

Where *B* is the eigenvectors and Λ is a diagonal matrix having the corresponding eigenvalues. Since *B* is orthogonal, its inverse is equal to its transpose. Thus,

$$\Sigma_T = \frac{1}{n-1} P^t (B \Lambda\ B^{-1})\ P \tag{2.31}$$

In order to reduce the variance, we can assume that *P* and *B* are equal. And since Σ_T is required to be diagonal and Λ is already diagonal, then,

$$\Sigma_T = \frac{1}{n-1} \Lambda \tag{2.32}$$

Finding the eigenvectors sometimes ends up with numerical errors. Singular Value Decomposition (SVD) is a matrix factorization method that prevents this issue. It factorizes matrices to three main components: left singular matrix *X*, right singular matrix *V* singular values matrix Σ. Factorization process takes place as follows:

$$X = U\ \Sigma V^t \tag{2.33}$$

By merging the last two equations, it can be observed that the produced form can be used in the calculation of PCA transformation matrix *P*.

$$X^t X = V \Sigma^t U^t U\ \Sigma V^t \tag{2.34}$$

Since *U* is a unity matrix, then its inverse is equal to its transpose as well:

$$X^t X = V \Sigma^t \Sigma V^t \tag{2.35}$$

And since $\Sigma = \sqrt{X^t X}$, then,

$$X^t X = V \Lambda V^t \tag{2.36}$$

By substitution, it can be declared that $B = V$. And since $P = B$ already and has a diagonal Σ_T, then $P = V$. To conclude, the output principle components start coming up with the highest variance variable (t_1) first and stepping down to the lowest variance one (t_p) [2]. The number of principle components for each multi-dimensional data sample can be pre-determined depending on the next application. For example, to plot all the multi-dimensional training samples on a 2D scatter, two principle components should be required. The process of reducing the dimensionality as mentioned in this section is called "Encoding". The reverse operation could be done as well in case if the principal components are available and the full-dimensional sample vector is required to be calculated. This operation is called "Decoding". Figure (2.42) and (2.43) illustrate the two phases of principal components analysis.

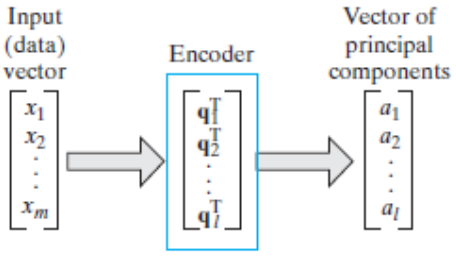


Figure (2.42) PCA Encoding [3]

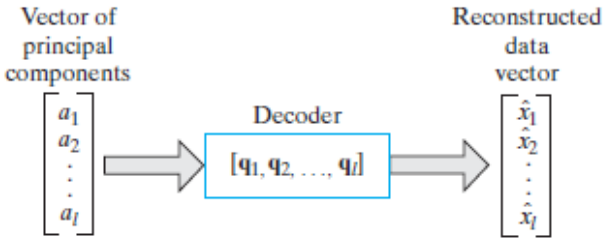


Figure (2.43) PCA Decoding [3]

2.10 Summary

This chapter reviewed and discussed fault detection and identification systems developed using machine learning techniques. Three types of learning are used in this thesis: Supervised, semi-supervised and unsupervised. In this chapter, the concepts pertaining to each of the three learning types were introduced and discussed. In supervised learning as applied to end of line testing of electric motors, when labels are available, it is easy for the system to understand the connection between the motor sample and its label, since the sample is already tagged with one of the motor fault types. Supervised learning has been implemented using artificial neural networks (ANNs) and support vector machines (SVM). However, in unsupervised learning, data samples are unlabeled, which requires mapping all data samples and, coming up with general rules that work for all samples. This is what is called the generalization step, which is the most difficult and uncertain process that occurs during the training process. Two different clustering algorithms were considered related to unsupervised learning: Flat clustering and hierarchical clustering. These methods are used for fault detection and diagnosis of electric motors. In semi-supervised learning, where only few samples are labeled, labels are required to enable unlabeled classification of samples. This can be achieved by using the Label Propagation (LP) algorithm.

Chapter 3

Supervised Learning

3.1 Introduction

This chapter shows the structure of the supervised learning algorithm used for fault detection and identification, as well as the results and discussions. An industrial dataset has been provided by "D&V Electronics" company. The data consists basically of sound and vibrations signals from test conducted on electric motors at the end of production line. It is important to study both sound and vibration signals together in order to judge whether the motor is healthy or faulty. Studying only one of them would give a wrong impression about the health status of the motor. M. Ismail, according to research reported in [2], extracted fault signatures from these signals using the Industrial Extended Multi Scale Principle Component Analysis [EMSPCA] method.

"Healthy" motors are the high quality products that are to be released to the automotive market while "faulty" motors are those which have one or more of the following faults:

1. Armature Fault. Wear in the carbon pads that supply current to the rotating armature is considered as an armature fault. Shorts or opens in the armature or field coils might be a reason as well and worn bushings might increase drag or allow the armature shaft to rub against the pole shoes. Any weak or broken part in the armature contacts would be included.
2. Brush Fault (i.e. worn brushes).
3. Planetary Gear Fault. Failures or unbalance in the planetary gear of the motor.

A healthy motor would not contain any of the above faults. This chapter focuses on building a supervised machine learning algorithm that can learn and figure out the health status of newly produced electric motors. The test cell overview and experimental setup of the IEMSPCA method are discussed in Appendix B.

3.2 Fault Signatures

M. Ismail [2] developed a Fault Detection and Diagnosis (FDD) solution for alternators and starters for its implementation in test cells. This solution extracted fault signatures from vibration and sound measurements using the Industrial Extended Multi Scale Principle Component Analysis (IEMSPCA), which is the core of the FDD strategy. In IEMSPCA, the raw sound and vibration signals were broken into time-frequency scales using wavelets to generate fault signatures. Two main stages were performed to produce fault signatures [2]:

- The Extended Multi Scale Principle Component Analysis (EMSPCA)
- Background Noise Elimination

Figure (3.1) shows an overview of the IEMSPCA process of the fault signatures from raw D&V Electronics dataset:

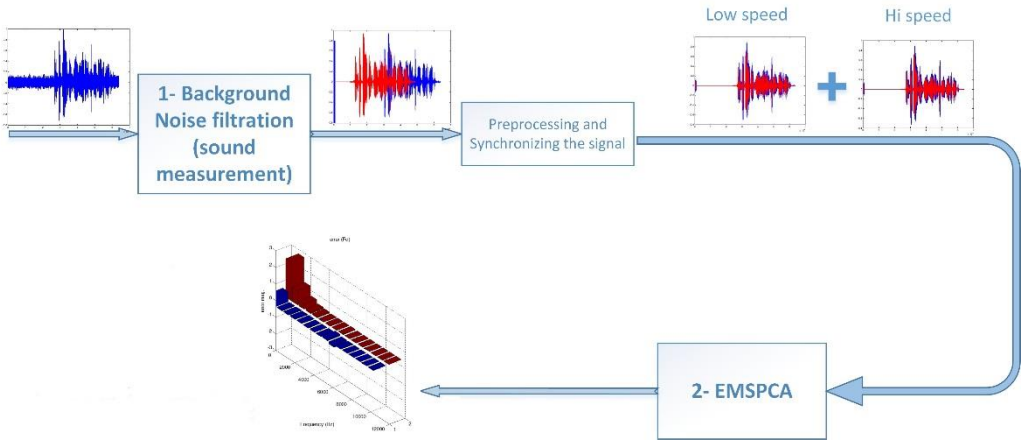


Figure (3.1) IEMSPCA overview [2]

Figure (3.2) shows the sound and vibration fault signatures of an alternator by applying IEMSPCA. The fault signature contains 2 measurements: Sound (i.e. maroon bars) and vibrations (e.g. blue bars). The vertical axis represents the error magnitude while the horizontal axis represents the frequency in KHz. Each measurement has 16 points describing the error magnitude at each frequency.

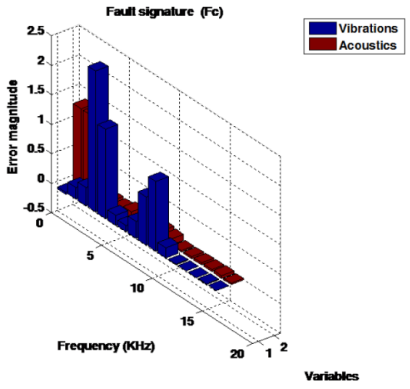


Fig (3.2) Alternator Fault Signature by EMSPCA [2]

3.2.1 Background Noise Filtration

This process was developed for an industrial production environment, thus the quality of the fault signatures was considered. Noise always exists in measurements obtained in industrial environments, and any background noise should be filtered before applying FDD algorithms.

The sound measurements provided by D&V Electronics were noisy. Noise gating was used to filter them using a threshold. When the real measurement was higher than background noise, the gate opened to maintain the output level as the input level. However, when the real measurement was not higher than the noise level, the gate was to close and attenuate the signal to remove the noise from the output. The signal was broken into two domains: Frequency and Time. That way, the filtration of the background noise could occur on separate frequency bins individually at each time segment. Figure (3.3) shows a flowchart of the noise gating process.

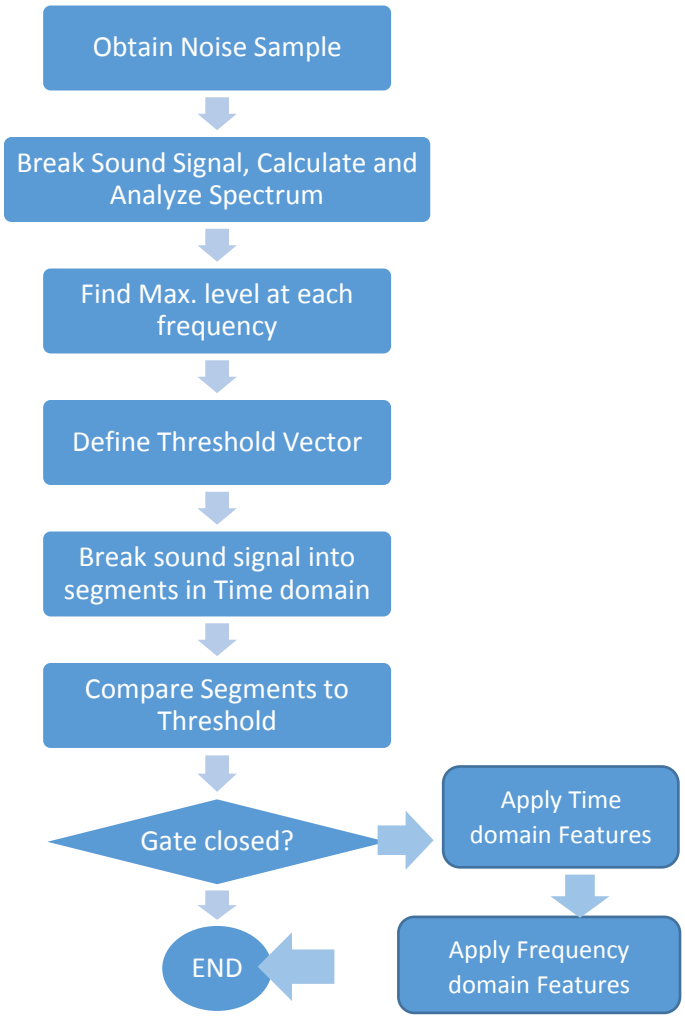


Figure (3.3) Background Noise Gating Flowchart

To improve the quality of the background noise isolations, different features were applied, such as frequency smoothing, attack time, hold time, release, hysteresis and attenuation range. The attack time was designed to slow down the noise gate when there are abrupt jumps in the sound signal. Furthermore, the hold time feature was used to hold the gate open for a little bit on purpose when abrupt sound closures exist. The attenuation range was used when the gate was closed. It defined how much attenuation is applied. Also, the hysteresis function controlled the gate to open only when sound

level hits the upper threshold limit in case of sound fluctuations. The process for noise isolation was performed as follows [2]:

1. Noise sample is obtained and cut off into small time domain windows.
2. Spectrum of each noise window was calculated using FFT.
3. Threshold vector was stored.
4. Sound signal was obtained and cut off into segments (i.e. similar to noise).
5. FFT frequency level of each segment was compared to threshold.
6. Attack, hold and release times were applied at frequencies lower than the threshold.
7. Attenuation Range and frequency smoothing were applied.

3.2.2 Extended Multi-Scale Principle Component Analysis

Extended Multi-Scale Principle Component Analysis (EMSPCA) was developed as a new strategy for the FDD methodology. A normalizing technique was implemented on the raw data to generalize the EMSPCA. This was to increase the accuracy of EMSPCA at different measurement types. The normalization was applied as follows:

$$Normalized\ Signal = \frac{Raw\ Signal - Mean\ Value\ (Baseline\ Center)}{Variance\ (Baseline\ Raw\ Signal)} \quad (3.1)$$

The signal was compared with the baseline as it is a signal-based algorithm. The diagram in figure (3.4) shows the order of different steps of EMSPCA algorithm.

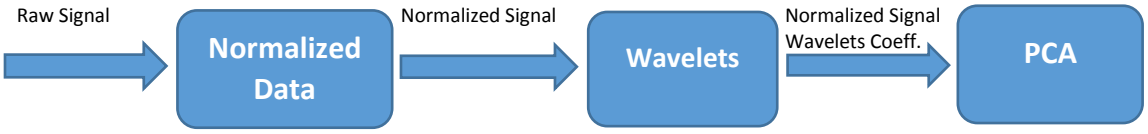


Figure (3.4) EMSPCA algorithm

A Wavelet Transformation Packet (WTP) (i.e. a wavelet transform where the discrete-time signal is passed through discrete wavelet transform (DWT) filter) was applied to the normalized signal. The wavelet packet was introduced in Multi-Scale Principle Component Analysis MSPCA (i.e. a sub-technique in M. Ismail's [2] research). It performed down sampling of the measurements by two and created shorter length wavelet coefficients at each scale. Thus, every second element in the raw measurements was eliminated. In the EMSPCA, wavelets were not fed directly to PCA as inputs, they were reconstructed using reconstruction filters. The measurements were then decomposed into a combination of scale based signals. The diagram in figure (3.5) describes the steps of applying WTP.

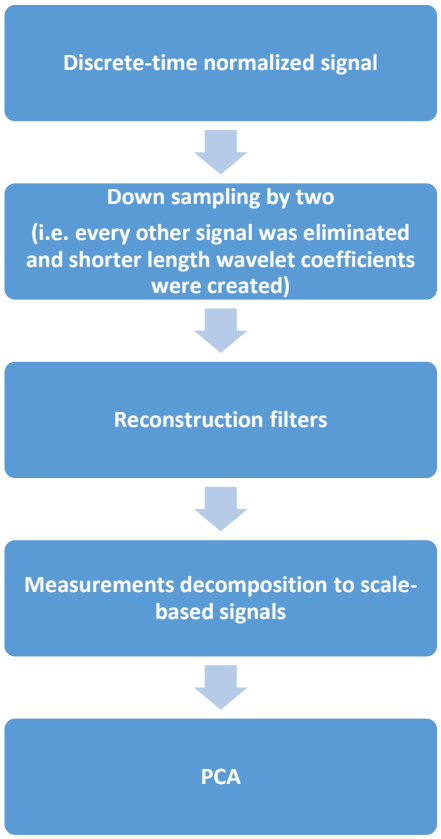


Fig (3.5) Wavelet Transformation Packet (WTP)

For J level wavelets decomposition, the output of the WTP was shown in equation (3.2) where X is the measurement, W_j is the DWT matrix, W_s is the synthesis wavelet matrix and G and H are low and high pass filters, respectively.

$$\begin{aligned} W_s W_J X &= H_1^t H_1 X + H_2^t H_2 X + ... + H_J^t H_J X + G_J^t G_J X \\ &= X_1 + X_2 + ... + X_J + X_{J+1} \end{aligned} \tag{3.2}$$

A special type of Principle Component Analysis (PCA), referred to as the correlation PCA, was applied then to the raw signal. Here, the mean value of the raw signals was subtracted, and the signal was then divided by its standard deviation given an input of normalized data. Figure (3.6) shows a summary of applying IEMSPCA.

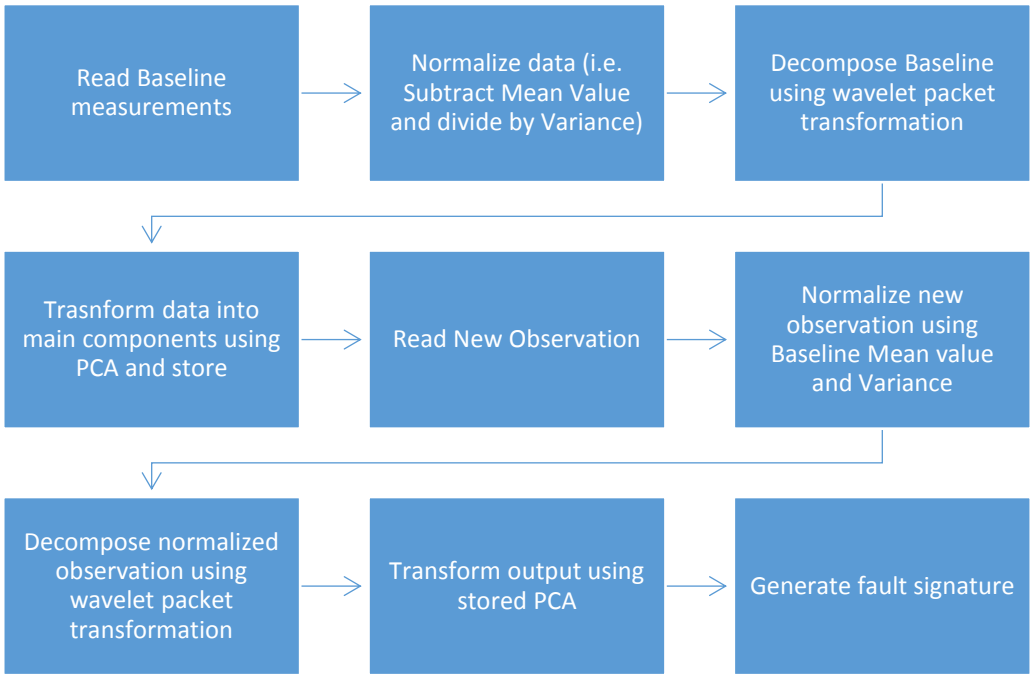


Fig (3.6) IEMSPCA algorithm Flowchart

The fault signatures generated by IEMSPCA method in [2] were then classified using a logistic classifier. The classification algorithm is discussed in Appendix C.

3.2.3 Summary of IEMSPCA

Two of the main advantages of this contribution that M. Ismail proposed in his research were background noise filtration and fault signature analysis. Figure (3.7) summarizes the entire IEMSPCA algorithm. The contribution was a complete Fault Diagnosis and Detection (FDD) system that was called Industrial Extended Multi-Scale Principle Component Analysis (IEMSPCA). Two main algorithms which were implemented to perform this contribution were: Background noise filtration and EMSPCA.

The system could be summarized in the following steps:

1. Obtain raw sound and vibration test measurements.
2. Filter background noise from sound signal.
3. Signal processing of the new observation (i.e. decompose signal into low and high speed test segments).
4. Apply EMSPCA to the output of (3) (i.e. generate fault signatures).

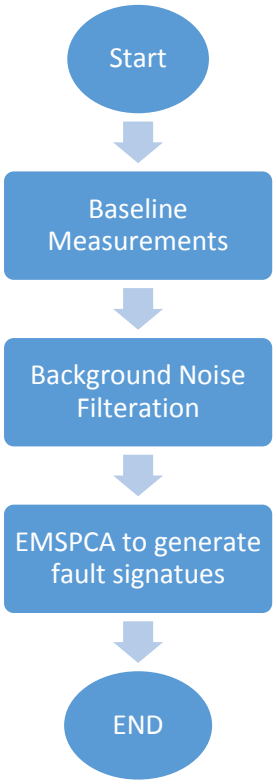


Fig (3.7) IEMSPCA summary

3.3 Case Study

The strategies considered in this research are applied to the case study used by M. Ismail [2]. This case study pertains to end of line testing of electric motor using a test cell developed by D&V Electronics. It benefits from a large database of test results collected by D&V Electronics. The raw dataset was collected using a high production starter tester "ST-118". The associated standard D&V test profile of this tester has two rotational speed profiles: low speed test at 1500 rpm and high speed test at 4200 rpm. Figure (3.8) shows the speed profiles. The dataset consists of 26,983 motor test samples.

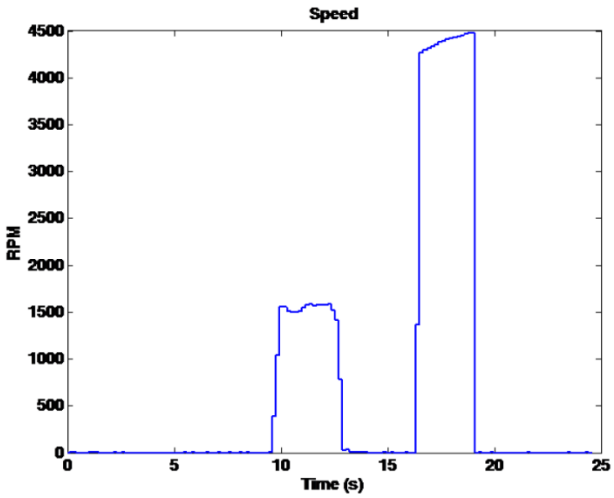
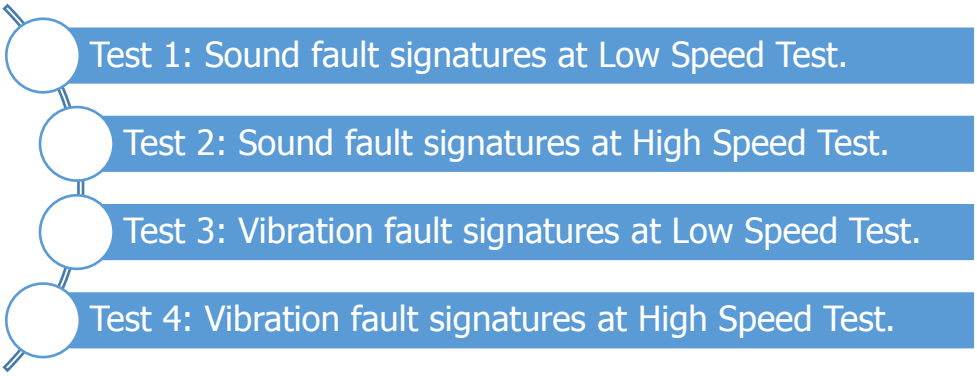


Fig (3.8) Test speed profiles for D&V tester ST-118 [2]

Four different tests have been done for each electric motor as follows:



Each test results one fault signature with 16D (i.e. dimensions/features/values). Thus, there is a total of 64 values for each motor. The first 16 dimensions (i.e. from Test 1) represent the acoustic features of a motor sample tested running at a speed of 1500 rpm, while the second test results those of the same motor running at 4200 rpm. Test 3 produces the fault signature of the vibration measurements at the low speed test, while the last fault signature represents those at a high speed test. Figure (3.9) shows the composition of the 64 dimensions from the four different tests.

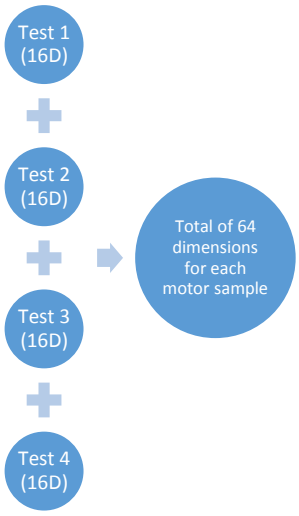


Fig (3.9) Composition of the 64 dimensions of each motor sample

The bar charts plotted in Figure (3.10) show examples of fault signatures from high speed tests. The difference among the different types of faults can be visually discriminated. A health status cannot be observed from one of them only, but the variations in both sound and vibration signals are integrative. Figure (3.10a) represents the fault signature of a healthy motor. The vertical axis shows the significant error magnitude values while the frequency in Hertz is plotted on the horizontal axis. It is noticeable that the error magnitudes are minimal at all frequencies, which means that there are none of the three fault conditions mentioned in section 3.1. Figure (3.10b) shows an armature imbalance fault signature where the error magnitude is most significant at the lowest frequency content while the vibration signature is almost flat. That is why studying both sound and vibration characters is important. Figures (3.10c & 3.10d), show the brush/commutator and planetary gear faults respectively, vibration contents are similar while the sound contents are discriminating features. Note that, the healthy sample (Fig 3.10a) and the planetary gear fault (Fig 3.10d), have almost the same error magnitudes in their acoustics, while vibration contents are totally different.

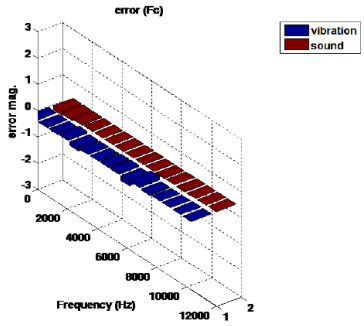
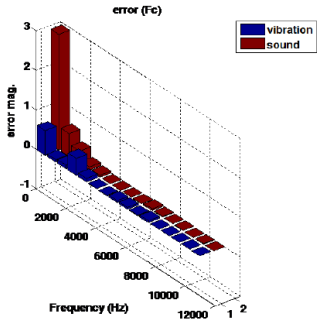
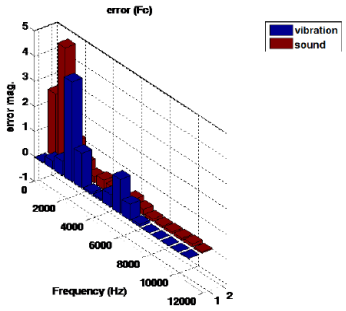
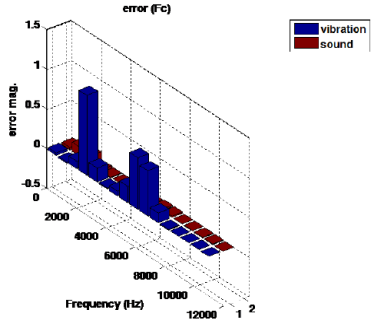
	Part type	Fault signature
a)	Healthy part	
b)	Armature imbalance fault	
c)	Brush\Commutator fault	
d)	Planetary gear fault	

Figure (3.10) Fault signatures of different motor fault types [2]

3.4 Supervised Learning Model Structure

Learning from labeled data requires a model that accommodates both training samples and their corresponding labels. This model is basically divided into two main processes: Training process and testing process.

The training process needs an amount of data that is enough to train the system, when a high performance is expected. This amount depends on multiple factors such as the quantity measured (e.g. sound & vibration) and its relevancy to the output predicted, the speed and detection rate of the test occurred, the complexity of the values and the dimensions of each sample. The sample features are then transformed from high-level features to low-level features in the form of feature vectors. The next step is clamping these features to the training model. Two training models have been implemented to achieve supervised learning: Artificial Neural Network (ANN) and Support Vector Machine (SVM). Those models will be discussed in details in section 3.4 and 3.5. The label of each sample is propagated in parallel with the low-level features of its corresponding sample. This is to learn which type of fault each fault signature belongs to, and thus learn how each fault type looks like in different motors and at different environments. Fig (3.11) shows a flow chart of data processing prior to the training models.

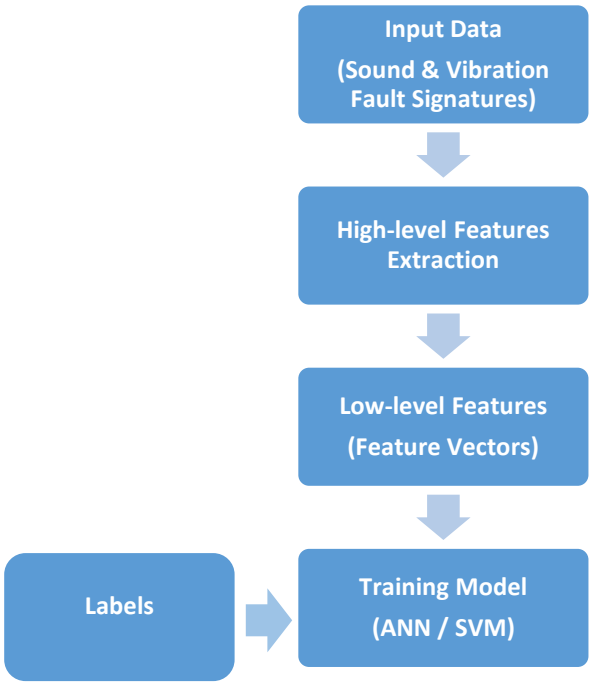


Fig (3.11) Data processing prior to Training Model

The testing process starts then with the predictive model that predicts what class (i.e. fault type) the new sample belongs to. It transforms down its features to low-level and looks up into all the low-level features that it has learned during the training process. The model ends when the label of the new sample is expected. Figure (3.12) shows the structure of the entire supervised learning model.

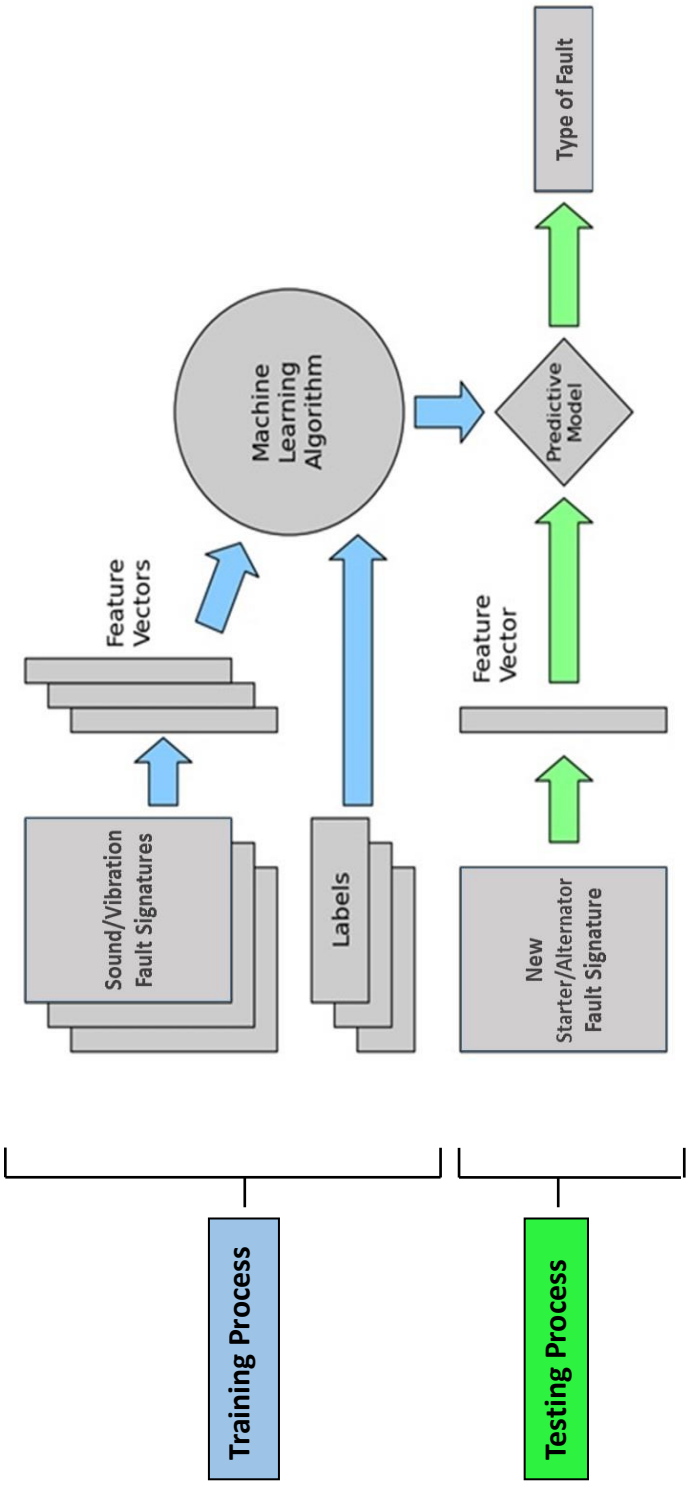


Figure (3.12) Supervised Learning Model Structure

3.5 Artificial Neural Network (ANN)

The first training model that has been implemented for supervised learning is the Artificial Neural Network (ANN). The network built for this project is similar to an auto-encoder, which is basically a neural network that produces a compact representation (encoding) of the input data, but in a supervised manner. The number of units in the hidden layers decrease in each additional hidden layer. That way, the network picks the most significant information from the input so that the output has the same contents but compressed in some way. For example, if the network is to recognize an input image of a cat or a car, it does not matter that the car is a jeep or a sedan, or red or black. It only matters that it is a car not a cat. In that case, some of the most significant information that an encoder would extract from the input data is that the image contains 4 wheels, hood, and mirrors. This is the only information that would be propagated to the next hidden layer. Similarly, in this model, the output of each layer in the ANN is the important information from the fault signature that emphasizes the fault type behavior.

3.5.1 Model Structure

Figure (3.13) shows the main construction of the ANN that was customized to train the system with the D&V dataset. This network consists basically of one input layer, two hidden layers and one output layer. The number of input units varies at every training process depending on how many fault signatures will be used from the four tests mentioned in section 3.3 for each sample. In data analysis, using all the fault signatures from different tests is not always helpful. It is important to train the network with each of the four tests outputs individually and compare their results to figure out which has clearer training data for the training process. Training the network with bad (i.e. unclear features)

fault signatures that are resulted from one of the four tests might mislead to a wrong classification at the validation (i.e. testing) process. The results will show that training the network with low speed test fault signatures individually leads to weak performance, unlike the results from the high speed test. The network training process can be done using only 16 dimensions of each the fault signatures or using any combination of those of the sound, vibration, low speed or high speed tests. Training the system with high speed tests is most recommended as it always gives higher detection rate than those at low speed tests. However, it is very important to train with both sound and vibration fault signatures as mentioned before, as they integrate together to present the health status of the electric motor.

At least one hidden layer is required to build up an ANN. This is where we get a few small nodes in the middle of the network to learn the data at a conceptual level, producing a compact representation of the input. That is why the number of hidden units must be less than that of the input layer. There is no rule to calculate the number of hidden layers required to train a neural network, it is a trial and error process. However, it highly depends on the application and the amount of data. Having a very compact representation of the input over one single step does not give the best image of the signal patterns. But extracting the best features step by step helps the system understand better these patterns. Thus, an application with higher amounts of data needs multiple hidden layers. However, adding more layers to a network that is trained on a small amount of data is not good as it would suffer from the problems of overfitting and vanishing gradients.

Finally, the output layer holds the target, which is one of the four types of faults numbered as follows:

- (0) means that the motor is healthy.

- (1) means that there is an armature imbalance.
- (2) means that there is a brush/commutator fault.
- (3) means that there is a planetary gear fault.

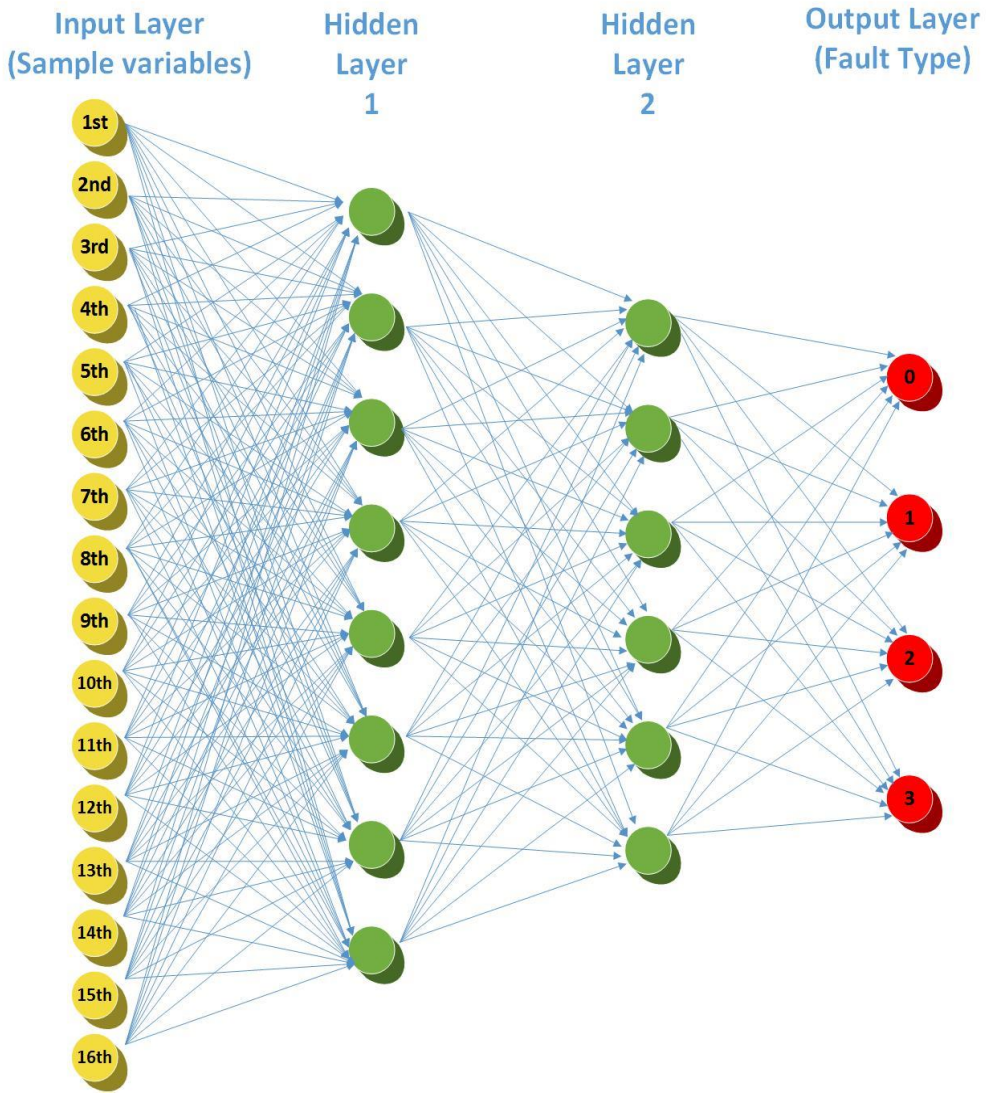


Figure (3.13) Neural Network Structure

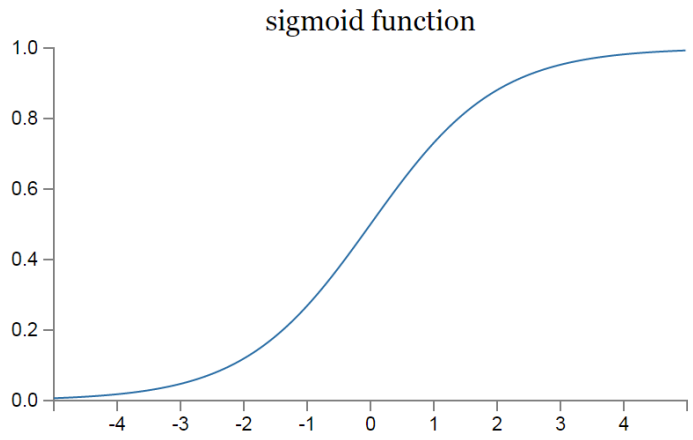
The input values are then propagated forward as inputs for the hidden units using the weighted sum transfer function for each hidden unit. The hidden units in turn calculate their outputs using the activation function. The output of each output neuron, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{3.3}$$

For simplification, two notational changes can be made to the condition. $\sum_j w_j x_j$ is basically a dot product [i.e. $w.x \equiv \sum_j w_j x_j$] where w and x are vectors whose components are the weights and inputs respectively. Also, moving the threshold to the other side of the inequality, and replacing it by bias, b $\equiv -$ threshold. Thus, the perceptron rule can be rewritten as:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{3.4}$$

The higher the bias of the perceptron is, the easier is to get 1 as output. It turns out that we can devise a learning algorithm which can automatically tune the weights and biases of a network of artificial neurons. This tuning happens in response to external stimuli, without direct intervention by a programmer [72-76]. In sigmoid neurons, a modified type of artificial neuron, small changes in their weights and bias cause only a small change in their output. Instead of being 0 or 1, they can take any values in between. Figure (3.14) represents the non-linear sigmoid function, which is actually the smoothed out curve of a step function.



$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

Figure (3.14) Non-linear Sigmoid Function

However, Δoutput is a linear function of the changes Δwj and Δb in the weights and bias. This linearity makes it easier to choose small changes in the weights and biases to achieve any desired small change in the output. So sigmoid neurons have much of the same qualitative behaviour as perceptron, and they make it much easier to figure out how the change in weights and biases will affect the output. Once defined, the network is trained using backpropagation method to update the weights using stochastic gradient descent technique. The result of the output layer is the output of the whole network [66].

3.5.2 Results of ANN

In supervised learning, some of the data are used for training and others are used for validation. The model learns the sample features from the labeled training set and a predictive model has been fitted to the 4 different types of faults (0, 1, 2, 3) to predict those of the validation set. This has been done by using the Python programming language using the Scikit-Learn library, which includes efficient tools for data mining and data analysis. Two properties have been studied: Sound and Vibration.

Figure (3.15) shows the confusion matrix (i.e. a table that describes the performance of a classification model) of the supervised learning model that has been trained with sound fault signatures at low speed test. This means that the network was fed with a 16-dimensional fault signature. The confusion matrix provides the distribution of the training samples and the validation samples for each of the different fault types as well as the average detection rate of class identification for the validation test.

As a start, the entire D&V dataset has been used for training except for 10 samples for validation. The average detection rate of fault identification was 82% for the validation test. It is remarkable that the number of samples is not equally distributed. The healthy motor samples represent about 53% of the total number of test samples provided, while the motors that have planetary gear faults are only 4.48%. This quantity imbalance is a big disadvantage in terms of data analysis. Giving the network different targets with equal number of samples would help increasing the performance of the whole model. Also, having only a quarter of the fault signature dimensions (16D). Furthermore, the low speed test does not reach all the high frequencies, but running the motor at 4200 rpm would give richer and clearer features of the sound signals. All the previous notes are reasons why a low detection rate was achieved.

Predicted Value		Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
	Healthy	96%	0	4%	0	14517	3
	Armature Fault	11%	89%	0	0	7137	3
	Brush/Comm. Fault	0	1%	82%	17%	4111	2
	Planetary Gear Fault	0	21%	18%	61%	1208	2
	Avg Detection Rate	82%				26973	10
		Actual Value					

Figure (3.15) Confusion Matrix of sound signatures at low speed test

Obviously, using sound fault signatures only did not give the desired performance, thus it is necessary to work on getting the highest possible detection rate in order to make the tester trust worthy. The vibration fault signature from the low speed test have been added to each sample on top of its sound fault signature. Thus, the model has been trained with 32 dimensions for each sample. The diagram in figure (3.16) shows the input data to the ANN.

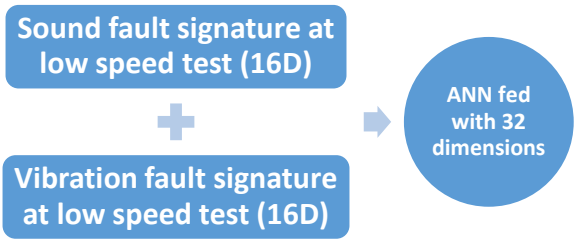


Fig (3.16) Composition of 32D input data to ANN

This addition has greatly improved the performance of the validation test to 96% as shown in figure (3.17).

Predicted Value	Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
Healthy	100%	0	0	0	14517	3
Armature Fault	0	100%	0	0	7137	3
Brush/Comm. Fault	0	1%	95%	5%	4111	2
Planetary Gear Fault	0	3%	6%	91%	1208	2
Avg Detection Rate	96%				26973	10
	Actual Value					

Figure (3.17) Confusion Matrix of sound and vibration signatures at low speed test

It was noticed that a remarkable improvement occurred in the performance of the model with the addition of vibration measurements at low speed test. This means that adding more features (e.g. dimensions) from other tests to the sample made it clearer and more efficient to learn from. Consequently, since the sound and vibration signatures of the high speed test are also available, all the 64 features (i.e. sound & vibration fault signatures at both high and low speed tests) have been added to the ANN. The following matrix in figure (3.18) shows the performance of the model using 26,973 64-dimensional samples for training and 10 samples for validation.

Predicted Value	Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
Healthy	100%	0	0	0	14517	3
Armature Fault	0	100%	0	0	7137	3
Brush/Comm. Fault	0	0	100%	0	4111	2
Planetary Gear Fault	0	0	0	100%	1208	2
Avg Detection Rate	100%				26973	10

Actual Value

Figure (3.18) Confusion Matrix of sound and vibration signatures at low and high speed tests

Obviously, the model has reached a 100% average detection rate and even for each of the individual fault types. This means that the network was very well studied and customized. There was no reason to decrease the number of training samples when the performance was not that good. But this model is being designed to fit multiple applications and different environments. Not all companies have large amount of data. Thus, the target should be building a supervised learning model that achieves the highest possible performance with the least possible number of labeled data. When the number of training samples was decreased, the model lost its high level performance.

This drop was expected because the more examples the model sees, the better mapping it creates. Some modifications and improvements had to be done to recover the same detection rate with less samples and the system parameters had to be tuned and optimized. The first hidden layer was set to 16 neurons while the second was set to 8. Also, since the network had more than 2 output neurons (i.e. 4 types of faults), it was categorized as a multi-class classifier. The softmax activation function is best used in multi-class classification instead of the logistic function in equation (2.2) which gives a binary output [10]. It increases the accuracy of the probability computations of each class, and is written as:

$$softmax\ (z)_i = \frac{e^{z_i}}{\sum_{l=1}^{k} e^{z_l}} \tag{3.5}$$

where z_i represents the *i*th element of the input to the softmax function. Furthermore, the training samples were equally distributed over the four types of faults. The output from equation (3.5) is the probability distribution of the four output neurons given the input value z_i. The sum of all the outputs (i.e. $\sum softmax\ (z)_i$) must be 1. The number of epochs has been increased to 300 epochs in order

to make sure the model has received all the information and learned them carefully. All these adjustments greatly improved the performance of the supervised model. The confusion matrix in figure (3.19) shows the results after the changes mentioned above. An average detection rate of 98% was achieved with only 400 labeled training samples, which represents 1.5% of D&V dataset. All the remainder of the dataset (i.e. 26583 samples) were used for validation.

Predicted Value	Healthy	100%	0	0	0	100	14421
	Armature Fault	0	97%	1%	2%	100	7039
	Brush/Comm. Fault	0	0	99%	1%	100	4013
	Planetary Gear Fault	3%	0	0	97%	100	1110
	Avg Detection Rate	98%				400	26583
		Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
		Actual Value					

Figure (3.19) Confusion Matrix of sound and vibration signatures at low and high speed tests

Figure (3.20) shows the variation of the average detection rate versus the number of labeled training samples for the ANN. The detection rate is still 100% but with only 800 training samples instead of 26,973.

The less the number of samples, the less the detection rate. The last concern about machine learning algorithms is the training and testing computation time. The supervised learning model is relatively fast compared to other types of learning. The whole process takes 5 seconds for both training and validation. The less the number of training and validation samples, the less the computation time.

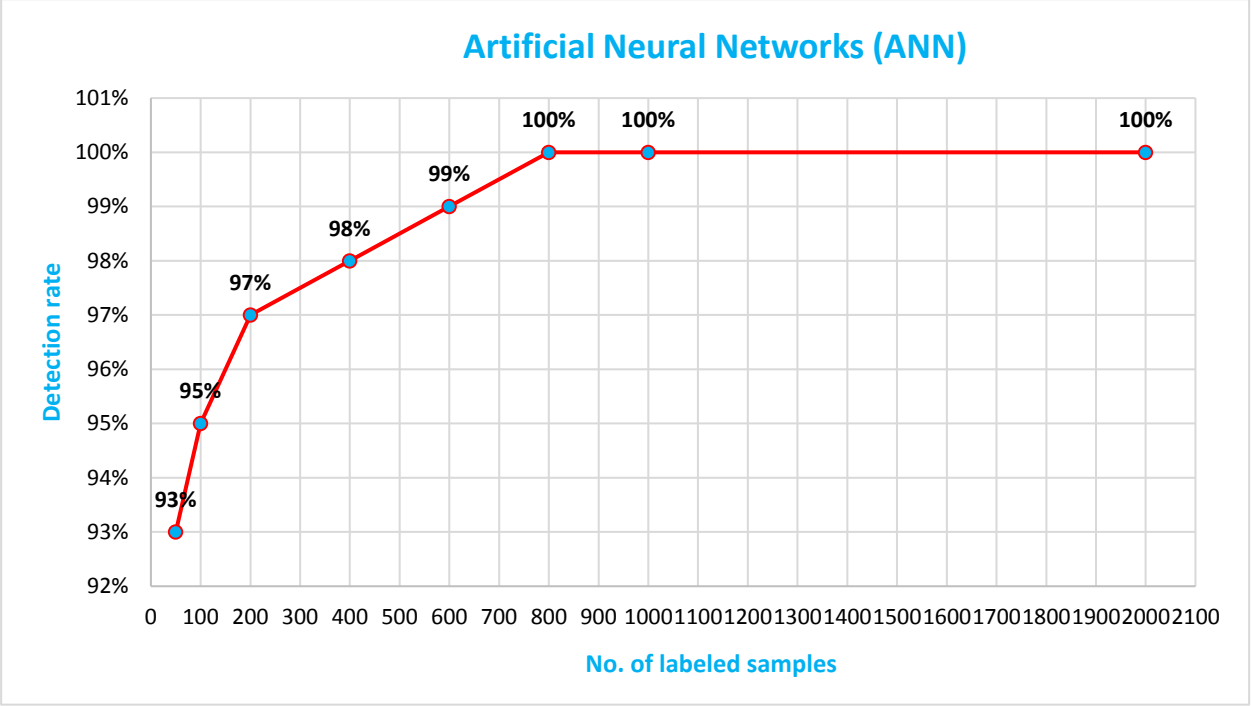


Figure (3.20) Variation of accuracies versus No. of training samples

3.6 Support Vector Machines (SVM)

As a supervised machine learning algorithm, support vector machines (SVM) can be used for both regression and classification challenges. But it is mostly used and recommended in classification problems. The training samples are plotted in an n-dimensional space, where n is the number of points (e.g. features) in each sample.

3.6.1 SVM Model

Support Vectors are the coordinates of individual observation, while the SVM is the hyperplane/line that best separates two classes. In the case of having more than two classes, a multi-class SVM should be created, which was the case in this project as there were 4 different classes. These classes are mutually exclusive (a.k.a. one-of, multi-national, polytomous, multi-class, or single-label classification), where each sample must belong to exactly one of the classes as shown in figure (3.21). If there are multiple faults in the same motor, the SVM nominates the fault that is most significant. To consider all possible faults at the same time in classification, a multi-class multi-label SVM is required. This was not possible to test in this project because D&V dataset was all single-labeled.

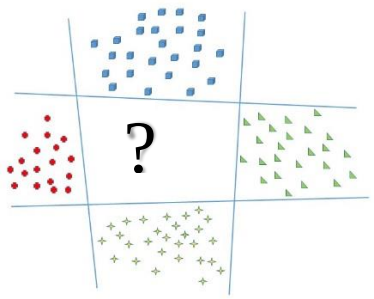


Figure (3.21) Support Vector Machines for 4 classes

The 4 hyperplanes which were designed for this problem do not divide the samples into distinct regions. Classes were ranked for each sample, then the top-ranked class was selected using equation (2.12). If the highest 2 classes are close to each other, only one class is selected as it is a single-label classification. Geometrically, the ranking is computed with respect to the distances from the linear hyperplanes. Misclassification is more probable when a sample is close to a class's separator. Thus, the greater the distance from the separator, the more probable is that the positive classification is correct. However, each class has a score, confidence value and probability. Eventually, each sample is assigned to the maximum score, the maximum confidence value or the maximum probability (see section 2.5.4.2).

3.6.2 Results of SVM

The results of the implementation of this algorithm would be best presented with confusion matrices as for neural networks. They show how many samples were incorrectly assigned to other classes. An average detection rate of 98% has been achieved with only 200 labeled training samples and 26783 validation test samples, which represents 0.7% of the dataset provided by D&V Electronics Company. The confusion matrix shown in figure (3.22) exhibits the detection rate and the corresponding number of samples used for training.

Predicted Value		Actual Value: Healthy	Actual Value: Armature Fault	Actual Value: Brush/Comm. Fault	Actual Value: Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
	Healthy	100%	0	0	0	50	14471
	Armature Fault	0	100%	0	0	50	7089
	Brush/Comm. Fault	0	4%	96%	0	50	4063
	Planetary Gear Fault	1%	0	2%	97%	50	1160
	Avg Detection Rate	98%				200	26783

Figure (3.22) Confusion Matrix of sound and vibration signatures at low and high speed tests

As shown in figure (3.23), the testing results change with the variation of number of training samples feeding the SVM. The detection rate reaches 100% at only 1000 training samples out of 26,973. The detection rate definitely drops down with the decrease in the number of training samples. The computation time of the SVMs is roughly 7 seconds. The less the training samples, the faster the results are obtained.

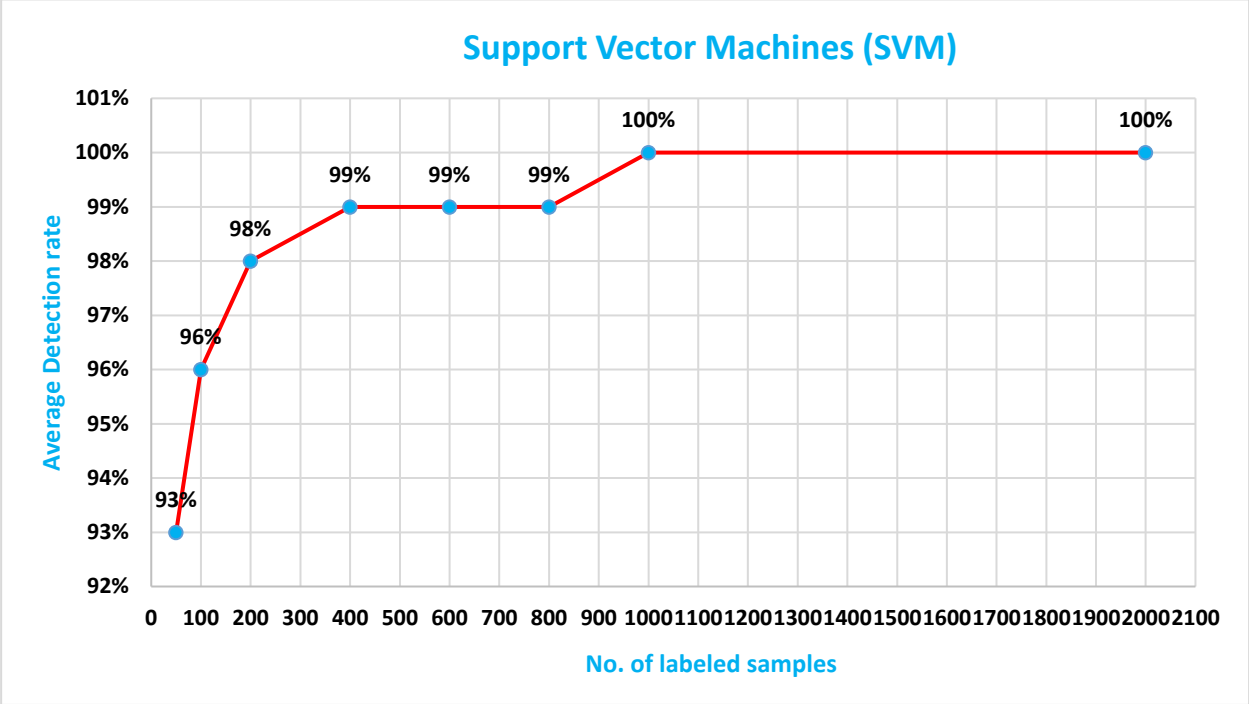


Figure (3.23) Variation of detection rate versus No. of training samples

The change in number of training epochs typically affects the performance of the training algorithm. As shown in figure (3.24), the average detection rate was 94% when the system started with only 10 epochs. It jumped up to 96% with more 40 epochs. The performance remains constant until 150 epochs are completed. The optimum performance that was achieved was 98% at 250 epochs. No changes occur to the performance with the increase of the number of epochs.

Time is a big concern for organisations which produce large amounts of motors, the computation time for end of production line test should be taken into consideration when increasing the number of epochs. This requires a fast training and testing algorithm. Figure (3.25) shows that the

computation time increases with more epochs. It takes 2 sec at 10 epochs and keep going up to 5 sec in order to run 300 epochs.

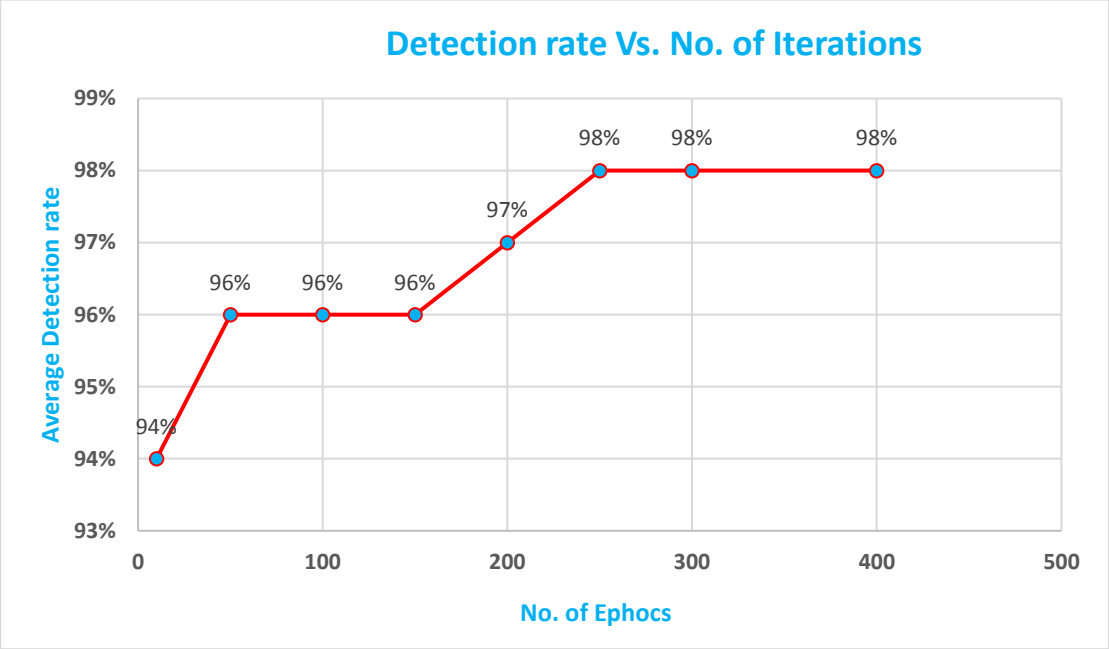


Figure (3.24) Variation of detection rate versus No. of Epochs

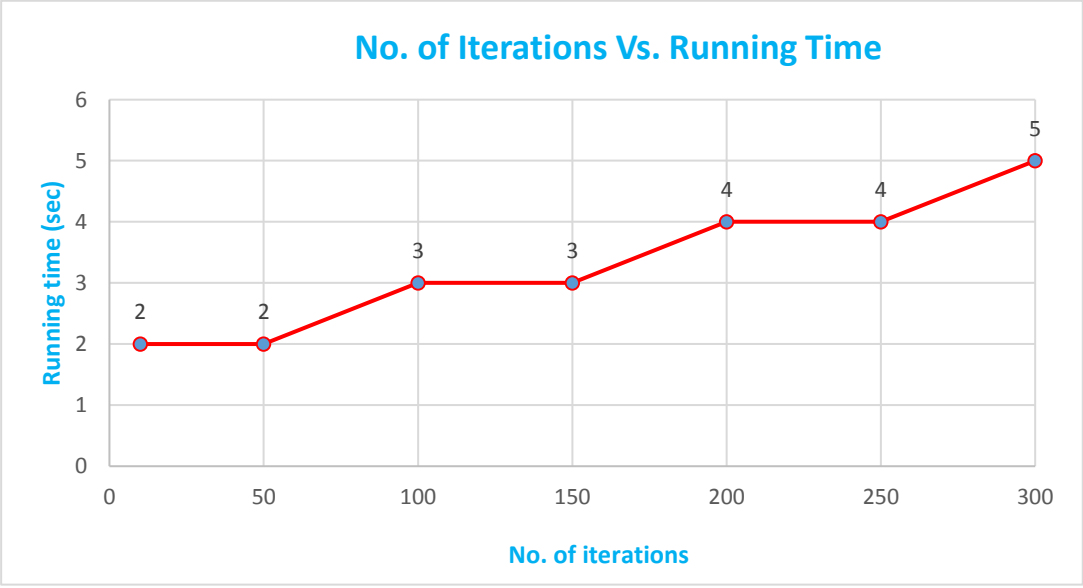


Figure (3.25) Variation of detection rate versus No. of training iterations

3.7 ANN or SVM ?

It is fair to admit that both artificial neural networks (ANNs) and support vector machines (SVMs) have performed well. Both of them reached high accuracies with very small numbers of training samples, but each of them has discriminating advantages. SVMs achieve 98% with only 200 training samples while ANN does that at 400 samples. On the other hand, ANN reaches a 100% detection rate with only 800 samples, while SVM makes it with 1000 samples. Thus, if a company has only few labeled samples, SVM would work better. However, if the detection rate of 100% is required with the least possible number of samples, ANN would be the better choice. Furthermore, time is an important factor for end of production line testing. Figure (3.26) shows the computational time for both SVM and ANN as being comparable.

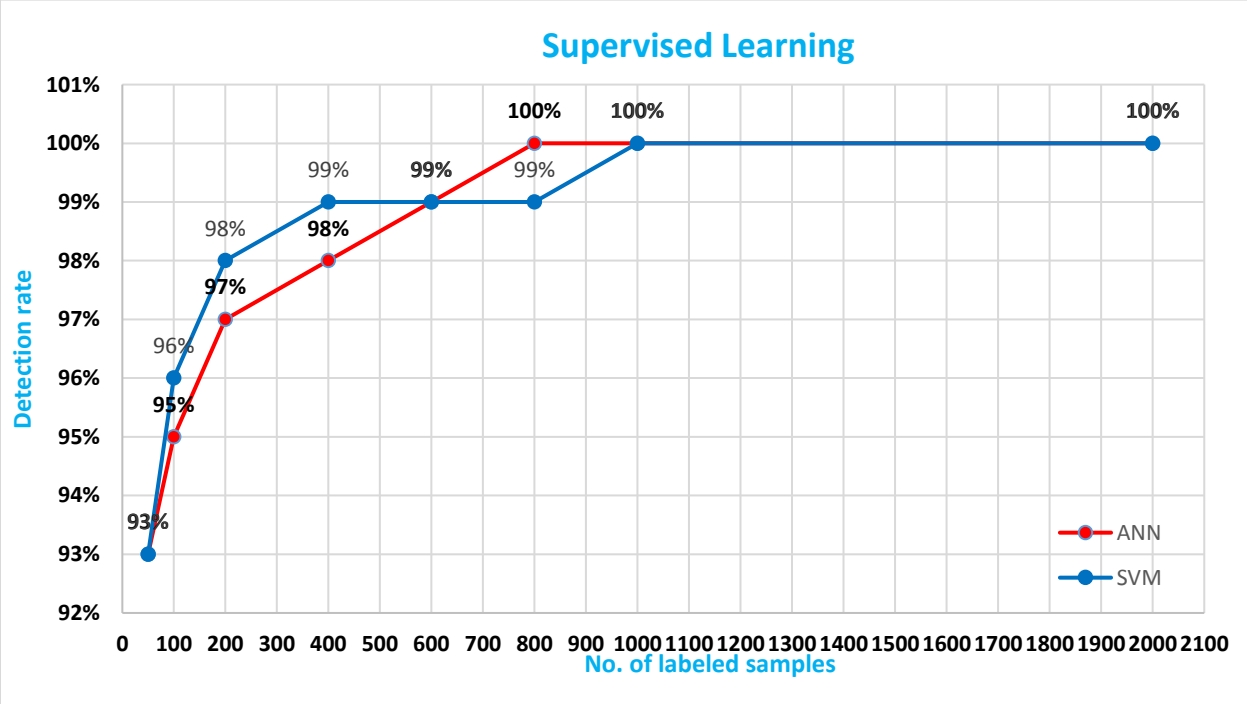


Figure (3.26) ANN results Vs SVM results

3.8 Diagnosis of New Untypical Fault Types

At the current point, the supervised learning model is capable of identifying the health status of a motor. The dataset provided by D&V Electronics does not include any samples having bad bearing or bent shaft faults. However, one of the objectives of this project, is to build a learning algorithm that is capable to detect previously unknown fault types. Thus, bad bearing and bent shaft faults would be detected but not diagnosed. To accumulate this objective, a model has been developed in this research that would diagnose a new fault type and warn the user about it, instead of just assigning it to the closest class.

The recognition of new untypical fault type has been realised with supervised learning. This was done by computing the probability of the new sample to belong to each of the 4 fault classes. The classifier is designed to assign the sample test results to one of the 4 classes. Two hundred sample tests have been studied with pre-known faults. The membership probabilities to the top ranked class were at least 99.97%, while the membership probabilities to the other 3 classes share the remaining 0.03%.

A probabilistic classifier was used to predict the probability distribution over the 4 output classes. It assigns a class label y to a sample x where $y=f(x)$. Assume X is the input dataset and Y is the class labels set of X. For a given $x \in X$, a probability $P_r\,(Y|X)$ is assigned to all $y \in Y$. Hard classification can then be done using the optimal decision rule:

$$Y = \arg\max_y P_r\,(Y|X) \tag{3.6}$$

Ten new uncommon fault signatures were invented. When they were tested, the probabilities that a sample belongs to the top ranked class were relatively much lower than 99.97%. Thus, the average

99.97% was used as a threshold for detecting a new fault is then diagnosed when its membership probability is lower than 99.97%, the sample is classified as having as a new untypical fault type. Here is an example in figure (3.27) of a new fault that was found.

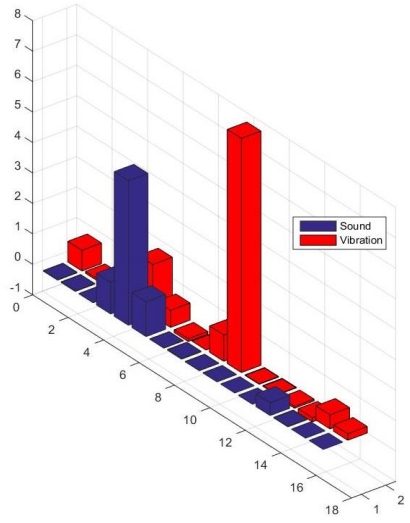


Figure (3.27) Example of an untypical fault

The fault signature shows a combination of Brush/Com fault and Planetary Gear fault which the model has never seen before. The probabilities calculated for each of the 4 classes are listed in table (3.1)

Fault Type	Probability
Healthy	0.00029%
Armature Fault	0.00017%
Brush/Comm. Fault	98.4%
Planetary Gear Fault	0.014%

Table (3.1) Variation of detection rate versus No. of training iterations

The classifier assigns the highest probability to the closest class, but since it did not exceed 99.97%, means that the system is uncertain and it does not belong to this class. The new fault has been involved into a set of 100 samples, and it was detected as the only unknown fault.

3.9 Summary

Chapter 3 discussed the structure and results of a supervised learning model. Two main supervised learning algorithms have been built, tested and compared. The first algorithm was the artificial neural network (ANN). It composed of one input layer, two hidden layers and one output layer. The network was trained using backpropagation algorithm (see section 2.2.3). The second algorithm was the Support Vector Machines (SVM). D&V dataset was used for training and testing the two learning algorithms. Both Artificial Neural Network (ANN) and Support Vector Machines (SVM) performed well. They reached high accuracies with relatively small numbers of training samples. SVM showed a detection rate of prediction of 98% with only 200 training samples while ANN reached this value at 400 samples. However, ANN achieved a 100% detection rate with only 800 samples, while SVM achieved it with 1000 samples.

Chapter 4

Semi-Supervised Learning

4.1 Introduction

The minimum number of labeled samples that could be used to obtain an acceptable detection rate (e.g. 98%) in the supervised learning model was 200 samples. The reason why a semi-supervised learning model is built is t the same job as the supervised model when there is a lack of labels. We learned in the last section that supervised learning is training the system with fully labeled data. Semi-supervised learning is best classified under the unsupervised learning category. It learns basically from a high number of unlabeled data in addition to few labeled data. The main intuition behind this combination is guidance.

4.2 Label Propagation Model Structure

The semi-supervised learning model extracts the features of the unlabeled samples, find similarities and differences between them, find relations between data and targets and generalize rules. There are several algorithms for semi-supervised learning. In this project, Label Propagation algorithm has been selected to be the learning algorithm for our model. It is basically a graph-based learning model that find communities within the dataset. The main idea is to label all the samples on a graph structure, when you are given few samples with labels. In comparison to other algorithms, Label Propagation has an advantage of short computation time and no parameter is required to be known beforehand. Figure (4.1) shows a sample of the Label Propagation model developed.

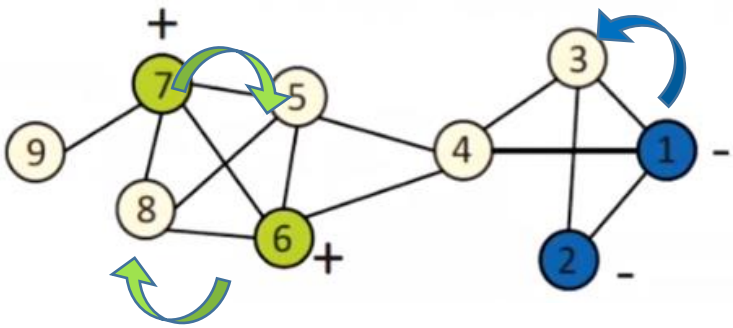


Figure (4.1) Label propagation Network

The dataset contains samples of four different classes, which are the four possible outputs: healthy, armature fault, brush/commutator fault or planetary gear fault. The system extracts first the features from all samples without paying attention to their labels. Then it matches the labels to their corresponding samples to build mappings between data and targets. The semi-supervised learning model structure is represented in figure (4.2)

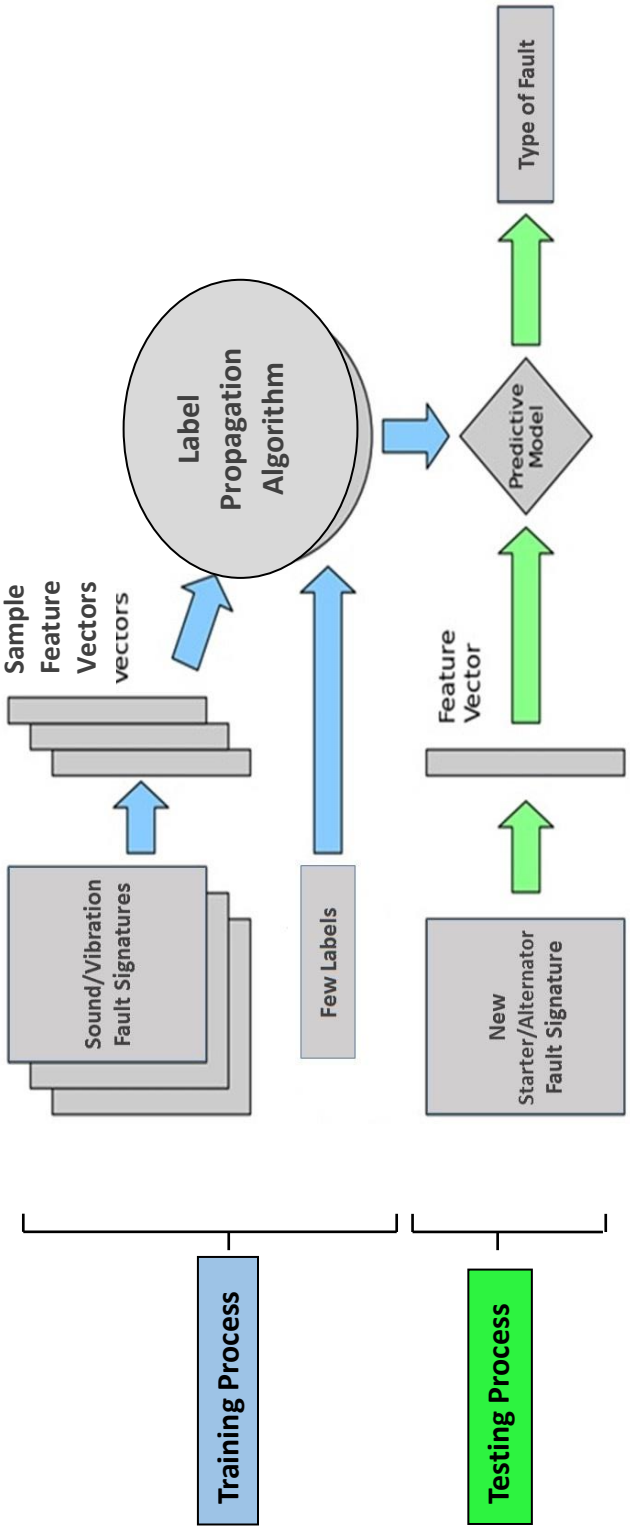


Figure (4.2) Semi-supervised Learning Model

Given few labels, the label propagation algorithm assigns every label to the closest unlabeled neighbor sample. The propagation continues in a random manner and the samples gather in groups of similar labels until all samples are labeled. Propagation stops once all samples are eventually labeled and groups of the same number of output classes (i.e. 4 fault types) are formed. Label Propagation algorithm was discussed in details in section 2.6.3.

4.3 Results of LP model

Unlike supervised learning, the performance of an algorithm that learns mainly from unlabeled data, with the guidance of very few number of labels, is expected to be relatively lower. The detection rate of the semi-supervised learning model using label propagation algorithm has begun the challenge with 56% for the validation test using 200 training samples (see Figure 4.3) at 100 epochs, including 20 labeled samples only and 180 unlabeled samples. The following confusion matrix shows the corresponding results.

Predicted Value		Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
	Healthy	94%	0	6%	0	45	14471
	Armature Fault	0	52%	0	48%	45	7089
	Brush/Comm. Fault	37%	13%	50%	0	45	4063
	Planetary Gear Fault	25%	19%	27%	29%	45	1160
	Avg Detection Rate	56%				180 (unlabeled) +20 (labeled)= 200	26783
		Actual Value					

Figure (4.3) Confusion Matrix of semi-supervised learning before modifications

To improve the detection rate, the system parameters have been tuned. The number of labels were increased to 30 labels and the number of unlabeled samples were increased to 370 samples. Furthermore, the number of training epochs was raised up to 300. The training samples have been also manipulated to the best training strategy. These changes have increased the detection rate of fault identification to 68% for the validation test as shown in the following figure (4.4).

Predicted Value		Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
	Healthy	96%	0	2%	2%	93	14421
	Armature Fault	15%	52%	0	26%	94	7039
	Brush/Comm. Fault	20%	0	63%	17%	93	4013
	Planetary Gear Fault	21%	3%	22%	54%	90	1110
	Avg Detection Rate	68%				370 (unlabeled) +30 (labeled)= 400	26583
		Actual Value					

Figure (4.4) Confusion Matrix of semi-supervised learning without Active Learning

Training the model with a dataset can be done with different strategies. In case of supervised learning, the choice of the training strategy is not as critical. But when dealing with unlabeled samples, Active Learning technique should be considered. Figure (4.5) shows the algorithms used for semi-supervised learning.

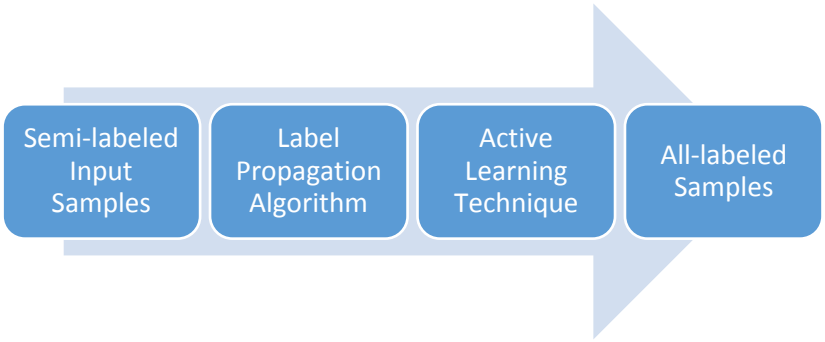


Fig (4.5) Semi-supervised learning process

4.4 Active Learning Technique

In the Active Learning technique, the learning process starts with a small number of labels as shown in figure (4.1), then the most uncertain points are selected to train next. In this project, Active Learning started with 10 labels that were selected according to the following: 4 healthy samples, 2 with armature fault, 2 with brush/commutator fault and 2 with planetary gear fault. The reason why a higher number of healthy training samples (i.e. four) is selected, is to avoid counting minor fluctuations in a healthy motor as faults. Using these 10 labels, the label propagation algorithm runs over all the unlabeled samples. The most 5 uncertain points are then selected to get labeled with their actual labels. Then, label propagation trains the system all over again with a total of 15 labels. This iteration is repeated until a learning exit criteria is achieved. However, the number of labels to add on each iteration, the number of iterations, as well as the total number of training labeled samples, are all adjustable if needed. A flow chart of Active Learning Technique is shown in figure (4.6).

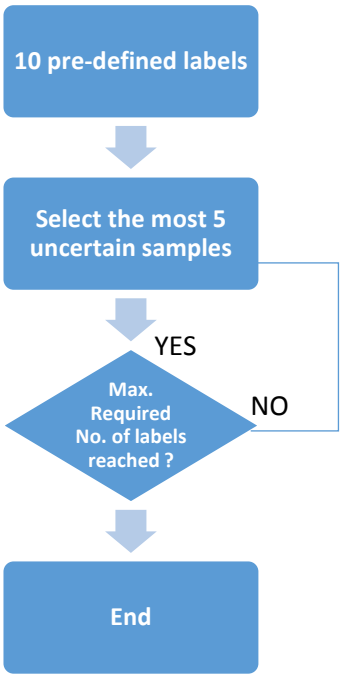


Fig (4.6) Semi-supervised learning Flowchart

4.5 Results of LP model with Active learning technique

The application of Active learning technique improved the performance to 83% for the validation test. This was mainly because of labeling the most 5 uncertain points at each iteration. However, the selection of the first 10 start-up labels was also important since random selection might choose 10 targets (e.g. labels) of a similar type of fault. Thus, equalizing the number of labels for each type of fault is important for initiating the training process. Shuffling the training set also improved the detection rate as the randomization avoids entire mini batches (sub-sets) of highly correlated examples, so each data point has an equal chance. Thus, randomized order of the data samples is strongly preferred rather than organized arrangement. If the training set is comprised of 400 observations, and the output is one of the 4 class labels (class I, II, III or IV), such that observations 1-100 are in class I, 101-200 in class II, 201-300 in class III and 301-400 in class IV, they should

not be presented to the network in that order. Figure (4.7) shows the confusion matrix after the implementation of the active learning technique.

Predicted Value		Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples	Number of Validation Test Samples
	Healthy	100%	0	0	0	93	14421
	Armature Fault	9%	69%	0	22%	94	7039
	Brush/Comm. Fault	8%	0	92%	0	93	4013
	Planetary Gear Fault	0	0	27%	73%	90	1110
	Avg Detection Rate	68%				370 (unlabeled) +30 (labeled)= 400	26583
		Actual Value					

Figure (4.7) Confusion Matrix of semi-supervised learning with Active Learning Technique

The following figure (4.8) shows the improvement in the detection rate by using active learning technique. It started by 10 labels and ended up with 30 labels with a step of 5 labels in each iteration. In comparison with supervised learning with the same number of labels (e.g. 50 labels), the detection rate was 93% while in semi-supervised learning it was 84%. This is because support vector machines and neural networks receive these labels as certain information that they can safely use to create rules from. In the semi-supervised model, the rules are made mainly from a large amount of uncertain

information (e.g. unlabeled samples) and the labels are just for guidance to start off the label propagation process. Thus, the model is less capable to generalize. The detection rate holds at 83% from 45 labels to 30 labels and slightly decreases to 82% at 25 labels. Then, it begins to drop down to 30% at 10 labels.

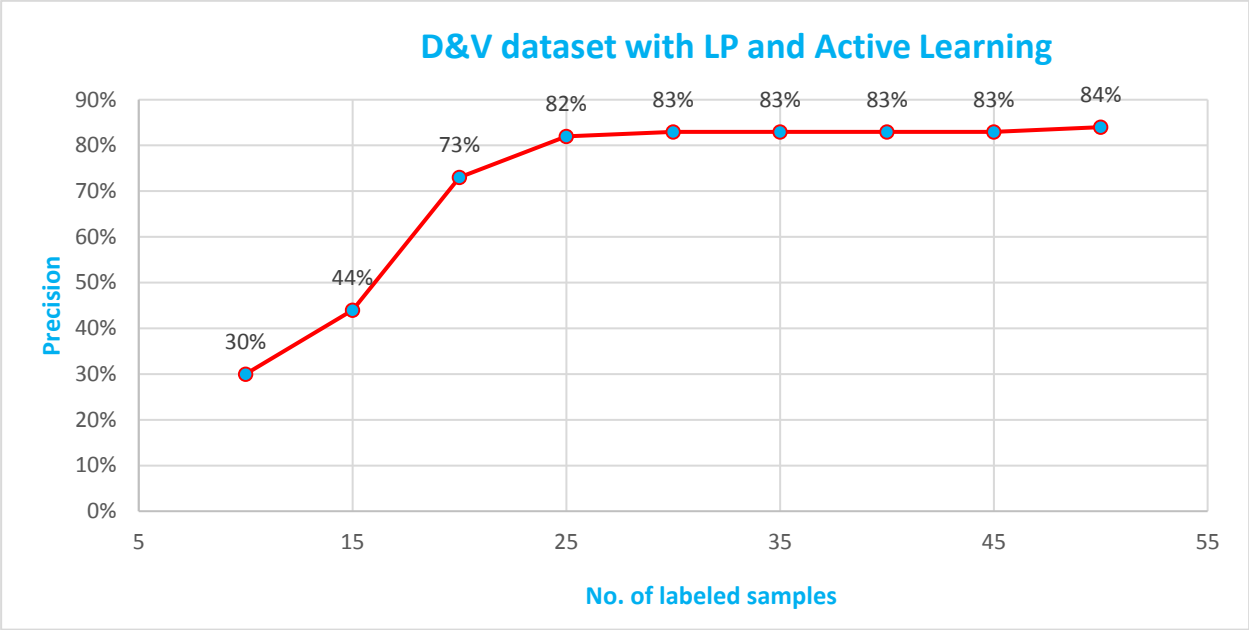


Figure (4.8) LP with Active Learning Technique at different number of samples

The training strategy of the unlabeled data also affects the performance of the label propagation training algorithm. Starting the training process with healthy motor samples is practically better than any of the other 3 faults. This is mainly because most of the healthy motor samples are very close to one another, while faulty samples vary in fault values. This makes it easier to create rules and generalize. For example, an armature imbalance fault signature typically has a high error magnitude in the earliest point of its sound fault signature. This spike would be in the first, second or third point (i.e. feature) of the fault signature and would be also high, very high or extremely high. This slight

dissimilarity decreases the capability of the learning algorithm to build mappings amongst unlabeled samples.

The semi-supervised learning model is reliable and applicable to other kinds of datasets. Let us consider and see the results of the same label propagation algorithm over the popular high-quality handwritten digits (MNIST) dataset (see figure 4.9). It consists of 784d (i.e. 784 dimensions) individual handwritten digits which represent numbers from 0 to 10. This dataset is very well studied and used to test machine learning algorithms.

Figure (4.9) MNIST Hand-written Digits [80]

The same model has been implemented over MNIST dataset and the results were relatively good. The detection rate resulting from this experiment was 92% with same number of labels (e.g. 30 labels) and 69,970 unlabeled samples.

However, getting higher results with MNIST dataset using the same algorithm and the same number of labels means that the algorithm is powerful enough, and there must be other reasons for the lower

performance with D&V dataset. Table (4.1) explains the main differences between the 2 datasets which noticeably affect the performance.

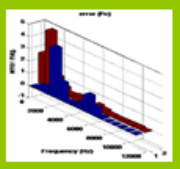
	MNIST Dataset	D&V Dataset
		
Data contents	Hand-written digits	Sound & Vib Fault Signatures
No. of Dimensions (per Sample)	784d	64d
Data Type	Large Integers	Small Decimals
No. of Samples	70,000	Less than 27,000

Table (4.1) Comparison between D&V and MNIST datasets

Each sample of D&V dataset has only 64 dimensions while MNIST dataset has 784 dimensions, which makes it richer with features. This large amount of features helps the learning algorithm to distinguish the samples and thus offers more guidance to identify the digit. The values of fault signatures are positive and negative small decimals which are relatively very close to one another while the handwritten digits contain only positive large integers (e.g. from 0 to 255). On the other hand, 70,000 training digits are available in the MNIST dataset [80] but only 26,983 fault signatures. All these advantages turned the use of handwritten digits more powerful than D&V dataset.

The active learning technique had a noticeable effect in raising the performance. The following graph shows the rate of increase of the performance during the implementation of Active Learning Technique.

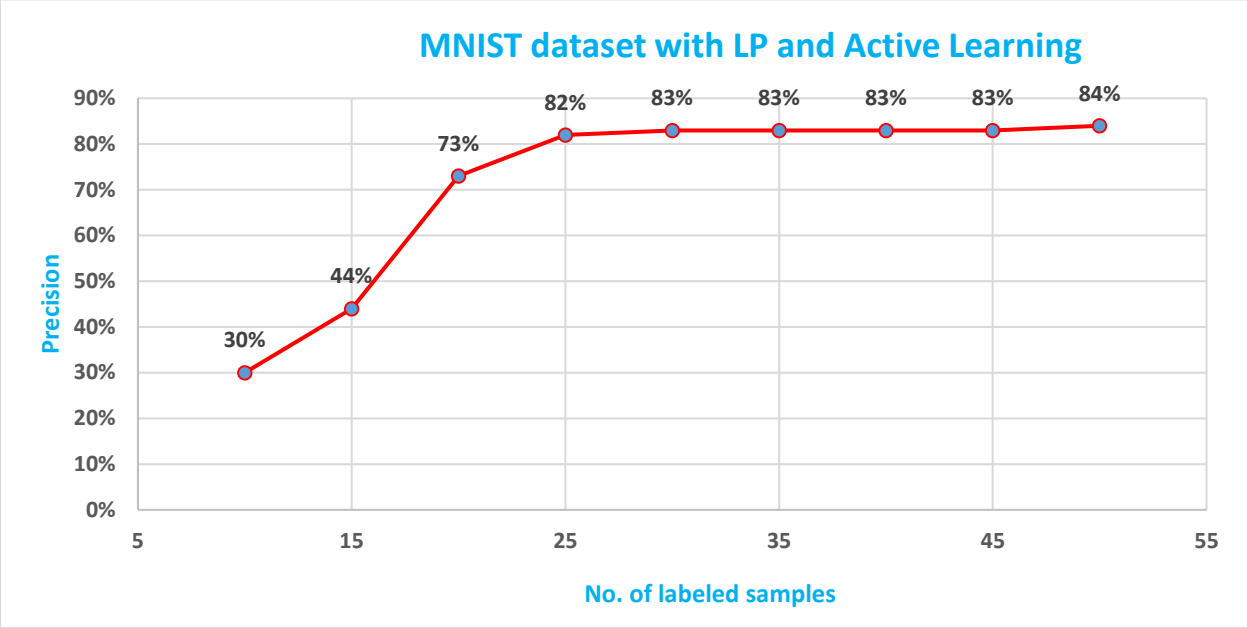


Figure (4.10) MNIST dataset average detection rate with LP and Active Learning

As shown in figure (4.10), a detection rate of 94% is achieved with 50 training labels, which is higher than the performance of the supervised learning using the same number of labels without any unlabeled samples. These results boost and support the objective of building a new type of learning algorithm with different capabilities. The detection rate slightly decreases to 91% at 25 labels, which is still relatively high. After that, it keeps dropping down smoothly to 52% at only 10 labels. This last value provides the motivation for exploring the development of if a machine learning algorithm that can learn from historical data with absolutely no labels. This is what will be explained in details in the next chapter which discusses unsupervised learning.

4.6 Summary

Labeling unlabeled samples given a large amount of unlabeled samples and few labels was the focus of this chapter. This was allowed by assigning each given label to the nearest unlabeled sample in a

graph-based training algorithm called Label propagation algorithm. This algorithm expanded the given labels to all unlabeled samples in a hierarchical way. Active Learning technique improved the performance of the semi-supervised learning model by labeling the most uncertain samples at each iteration. That way, error accumulation was avoided and the detection rate was improved. The maximum number of labels used for the label propagation process was 30 labels with a step of 5 labels in each iteration. The detection rate remained 83% from 45 labels to 30 labels and 370 unlabeled samples and decreased to 82% at 25 labels. Then, it drops down to 30% with only 10 labels. Semi-supervised is theoretically the best choice among the 3 types of learning. It gives a competitive detection rate with a minimal amount of labels, but it needs more unlabeled data to give the best possible detection rate.

Chapter 5

Unsupervised Learning

5.1 Introduction

Labeled data are not always available. Some companies and organizations have huge amounts of data but they do not have their labels. Real-time applications are always unlabeled as well, where objects are detected but not differentiated. Since the samples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. The challenge is how to benefit from these data and use it to identify similar samples in the future. Unsupervised learning studies how systems can learn from unlabeled data by mapping all samples and make up rules and generalize.

In this chapter, the dataset provided from D&V Electronics will be studied regardless of the labels. The unsupervised learning algorithm will learn how to identify different types of faults for end of production line testing of new motor samples. Here, the system does not need to know in how many clusters the dataset should be divided to. It studies the features of each sample and classifies the

different features. Two different clustering algorithms have been studied so far in this project: K-Means (i.e. Flat Clustering) and Mean-shift (i.e. Hierarchical Clustering).

5.2 Model Structure

The unsupervised learning model was built similarly the supervised learning model but with a total removal of labels. Figure (5.1) shows the structure of the unsupervised learning model of this research.

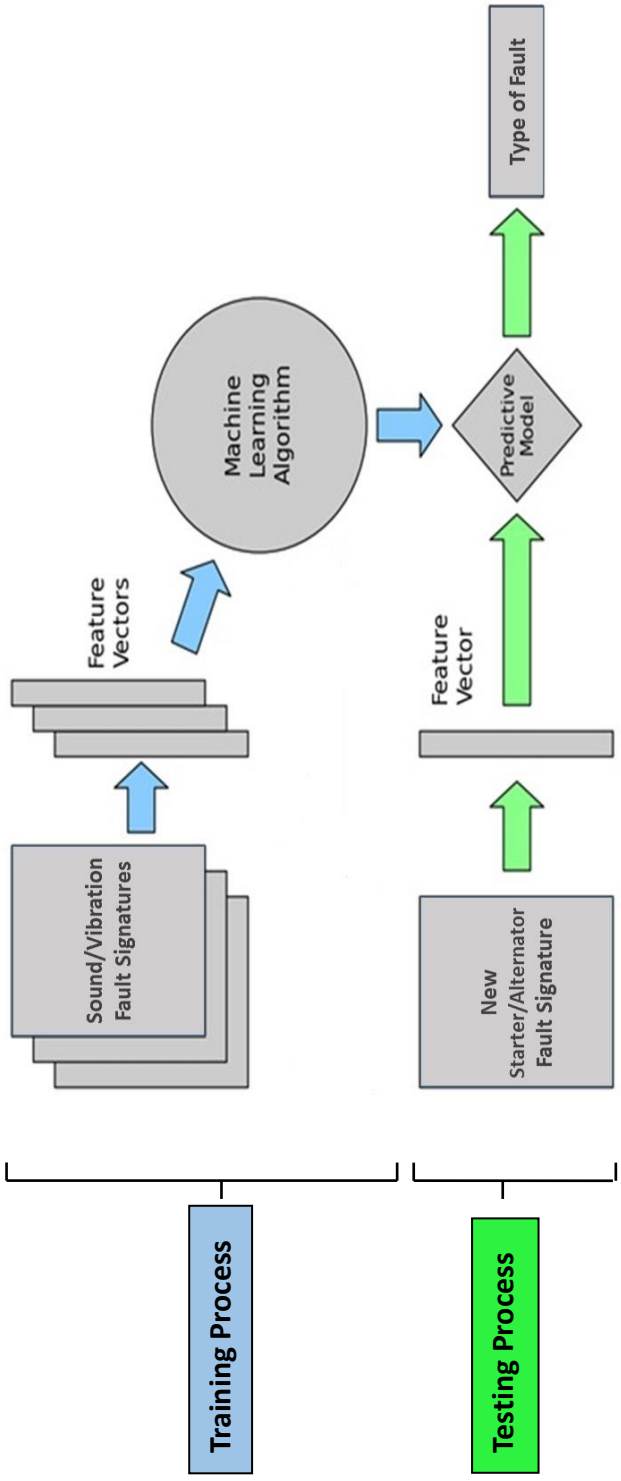


Figure (5.1) Unsupervised Learning Model Structure

5.3 Flat Clustering

K-Means is a clustering algorithm that keeps applying two main steps continuously until it reaches the no-change level. The first step is the "Cluster Assignment" where random centroids are selected and each point is classified to the nearest centroid. The number of centroids can be predetermined to the system or not. The second step is "Move Centroid" where the centroid, selected in the first step, is moved to the average point of all the classified points of the corresponding cluster. The two steps should be repeated until no further change happens.

5.3.1 Principle Component Analysis

A common barrier in visualising the clustering results is the high dimensionality. D&V dataset is 64-dimensional, which is impossible to be plotted on a 2D or a 3D scatter. There are two clues to deal with this type of data. First, is to run the clustering process over the dataset without visualising the results. Thus, human observations will not be allowed anymore to take further actions, only numbers would be used from the numerical analysis. The other option is to reduce the dimensionality of the data samples to 2 or 3 dimensions that can be visually available. But the penalty will be in the information content hence affecting the performance. This is due to the ability of the feature extractor to collect much information about the sample. Principle Component Analysis (PCA) is a technique used to examine variation for the purpose of dimensionality reduction (See discussion in section 2.9.1). This is done sometimes for a better visual representation as in this project or to reduce the processing time (i.e. memory). The output of PCA brings out the strongest patterns that best represent the observed variables in a dataset. The output of PCA do not actually have any physical quantities, they are simply the weighted combinations of the original variables of the samples.

All fault signatures (i.e. Sound, Vibration at low speed and high speed tests) of the 26,983 samples were reduced from 64 components to 2 components using PCA. Principle Components basically find the directions of the maximum variances of the sample components. The weight of each component varies at each principle component, based on the density of the components at this direction (See Fig 2.41). The numerical values of the Principle Components of each sample were calculated as follows:

$$PC = (x_1\ w_{1n}) + (x_2\ w_{2n}) + \ldots + (x_{64}\ w_{64n}) \quad (5.1)$$

where x is the value of the original component of the sample, w is the weight of x at the n^{th} Principle Component. Further details about the implementation of Principle Component Analysis were discussed in section (2.9.1). Fig (5.2) shows a flow chart of the computation of PCA.

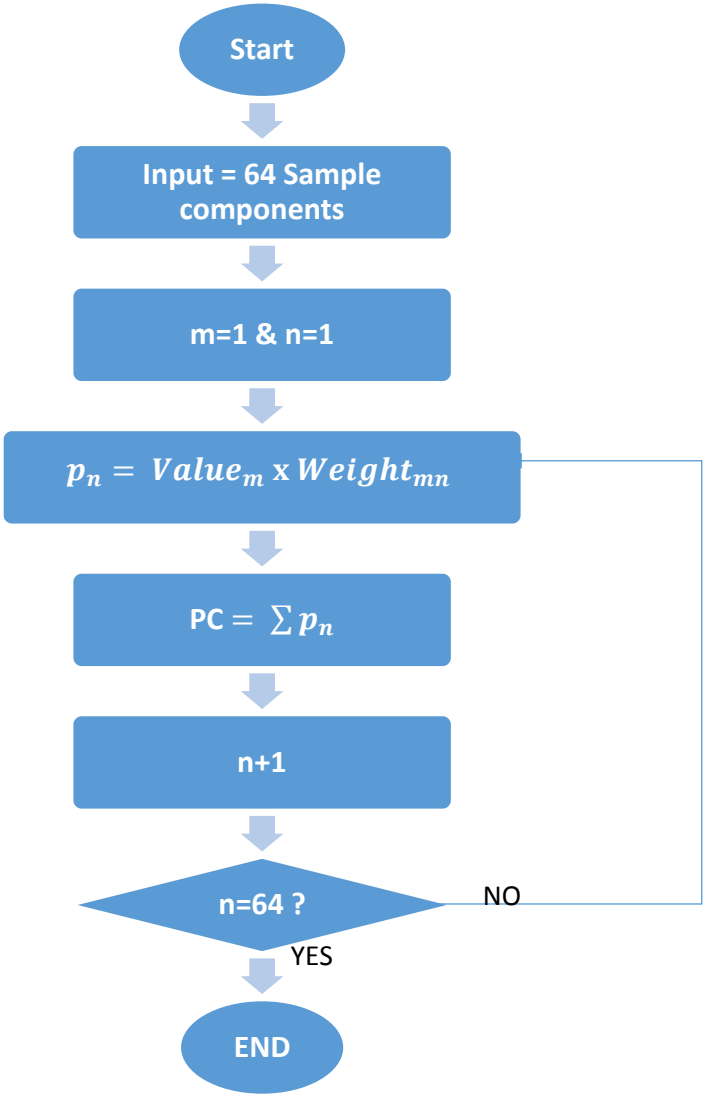


Figure (5.2) PCA Computation

The output was plotted on a scatter in 2-dimensional points as shown in figure (5.3). Each point represents the most distinguished features of a single sample. The horizontal axis represents the first principle component "PC1" while the vertical axis represents the second principle component "PC2".

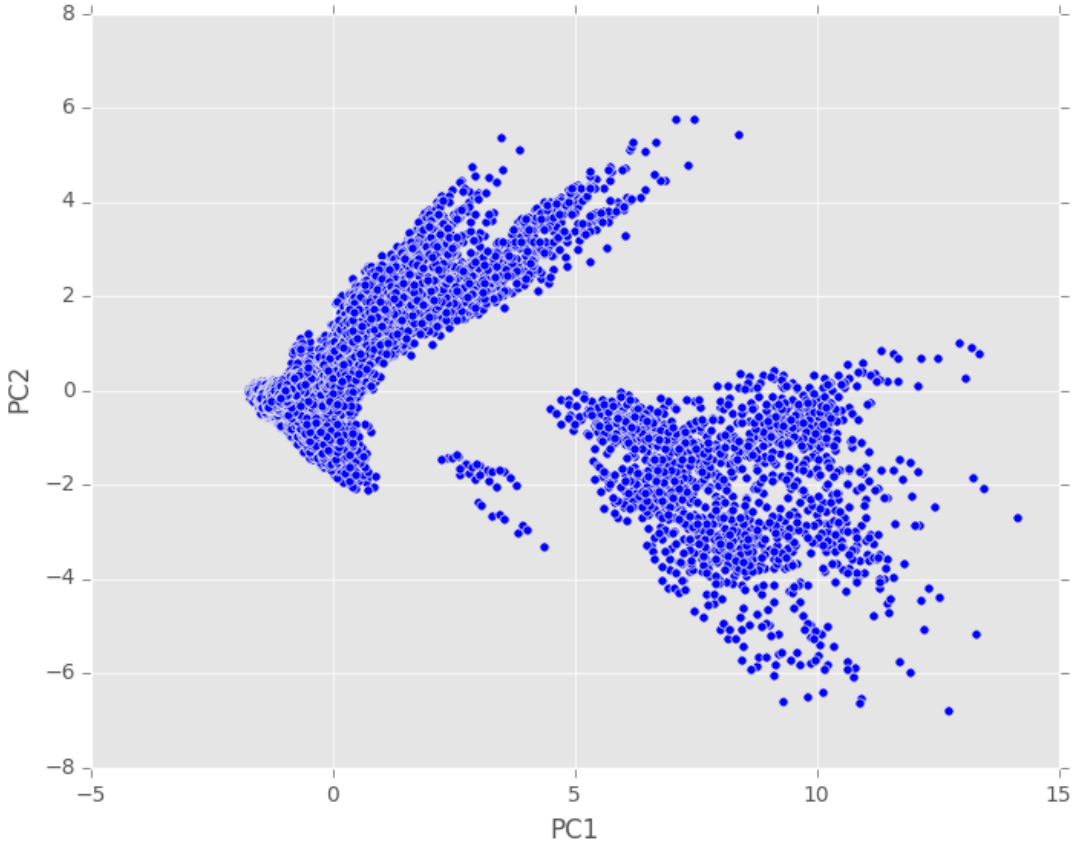


Figure (5.3) D&V dataset in 2D after PCA

The computation time to plot Principle Component Analysis is ~ 7 sec. The Principle Components of all samples have been then fitted to K-Means clustering algorithm (See discussion and example in section 2.8.1). Four clusters were pre-determined to the system by the user while the centroids were generated by the algorithm. Figure (5.4) shows a flowchart of the clustering process using K-Means algorithm.

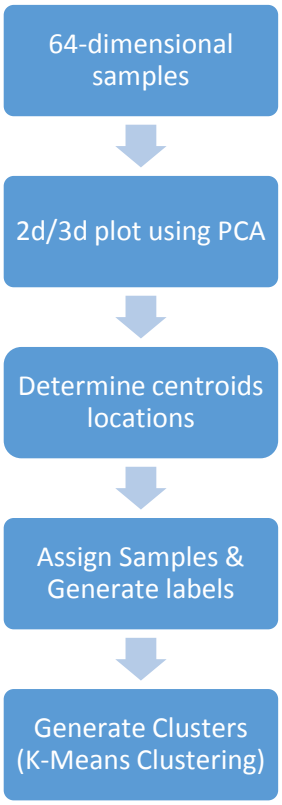


Figure (5.4) K-Means Clustering Process

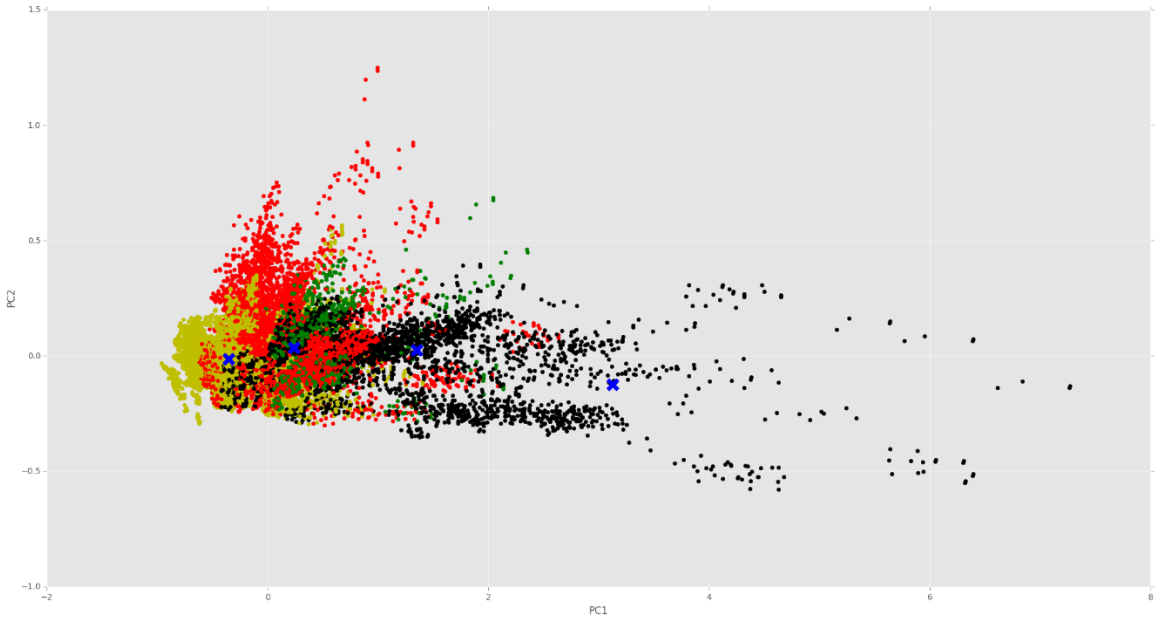


Figure (5.5) Labeled Sound fault signatures at low speed tests on 2D scatter

Figure (5.5) represents the actual scatter of pre-labeled data. The blue "x" signs are the centroids of each class. However, these centroids have been computationally generated during the clustering process. Flat clustering has been tested using the same data samples but in an unsupervised manner (i.e. unlabeled data). Here the number of clusters has been determined but the labels were generated by K-Means algorithm in an unsupervised manner. Figure (5.6) shows the scatter representing the unlabeled data clustered by this clustering algorithm.

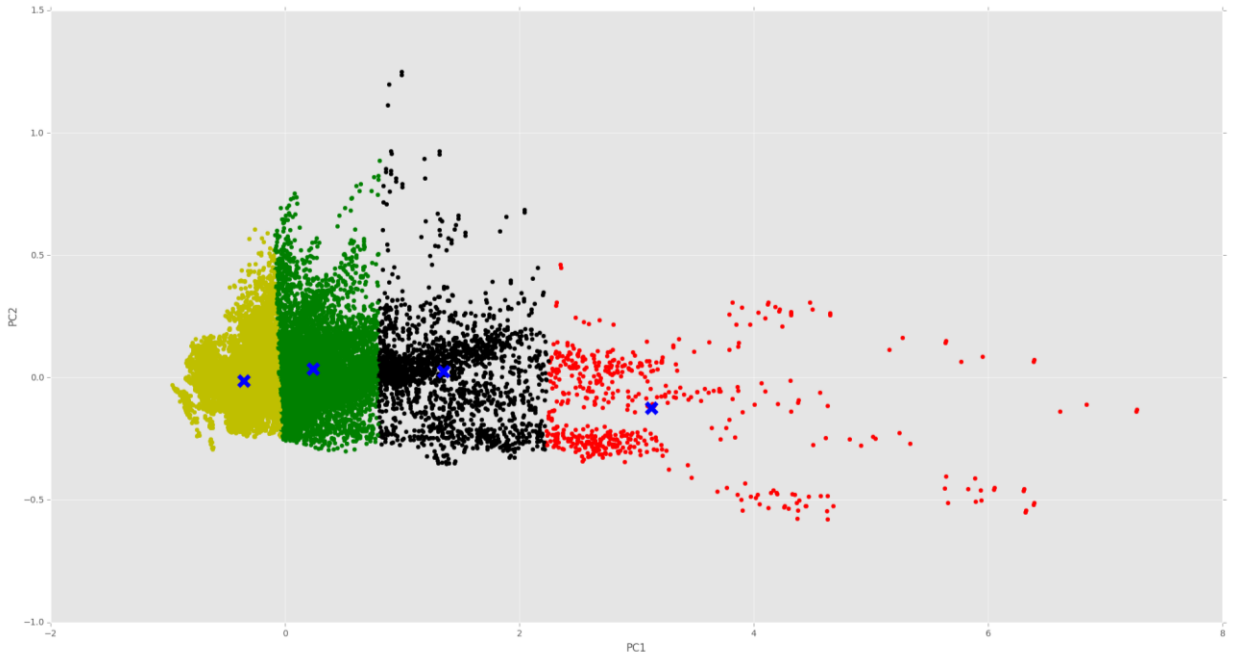


Figure (5.6) 2D Flat Clustering of unlabeled Sound fault signatures at low speed tests

The unsupervised clustering results are obviously not matching with the supervised ones. That is because there are several criteria that must be satisfied to cluster the data. It is impossible to find out a rule that selects the best clustering algorithm since no guiding information is given about the input data. Having the same example of the cat and the car images, if some of the images include white

cats and white cars, both might be clustered to the same group because the clustering algorithm is taking the color as a distinguishing tool. A car having a catty vinyl or cover might be clustered as a cat. Also, the cat's eyes in an image might be interpreted as vehicle front light beams. Similarly, sound and vibration fault signatures of some types of fault might be close in error magnitude, frequency, rate of increase or even location of spikes. One of the most important factors for picking the best clustering algorithm is the amount of data. The more data the algorithm receives, the better it is able to find differences between samples.

The colors in figure (5.6) show clearly that the data samples are clustered almost vertically upon the value of the first principle component (PC1) regardless of the second principle component (PC2). The first cluster starts from -0.9 to 0 representing label (0) while the green one moves on to 0.8 for label (1). Then, label (2) covers all the data from 0.8 to 2.3 and the rest of the samples in red are clustered as label (3). Furthermore, flat clustering has been applied over high speed tests data as well as shown in figure (5.7).

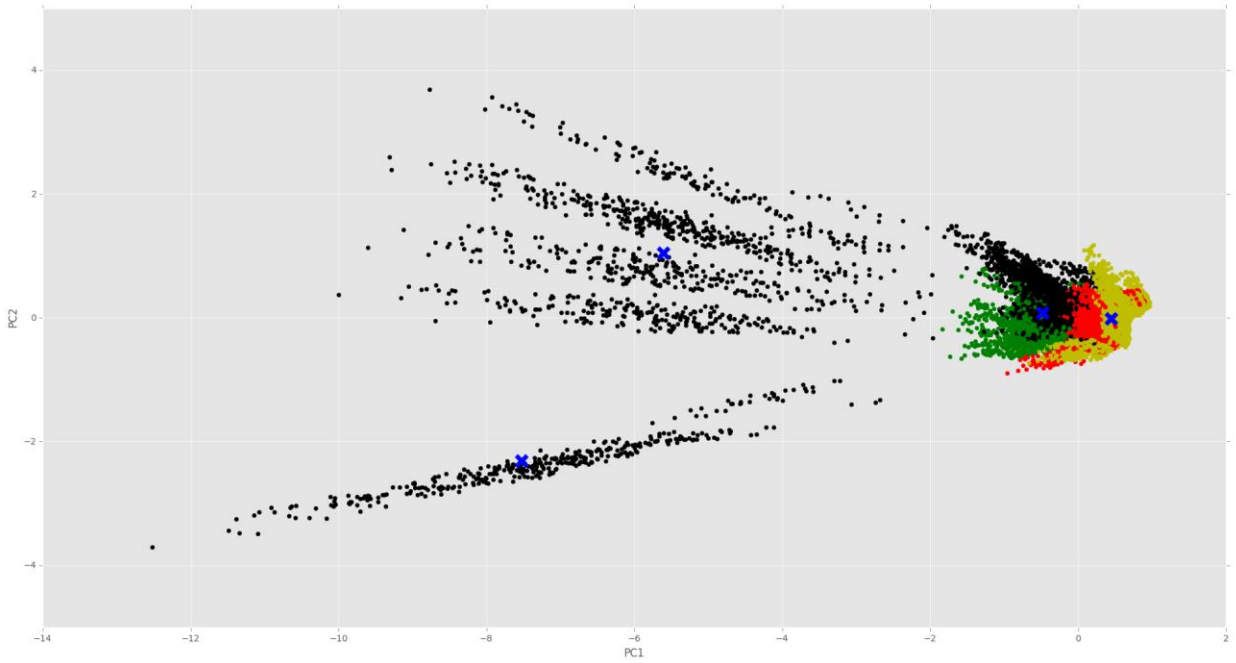


Figure (5.7) Labeled Sound fault signatures at high speed tests on 2D scatter

The labeled representation exhibits the healthy motors in black, while the armature faults are colored in yellow. The brush/commutator faults are in classified as the red samples and on the other hand, the fewest number of samples in green are the planetary gear faults. Despite this, Figure (5.8) shows the results of K-Means algorithm on this data.

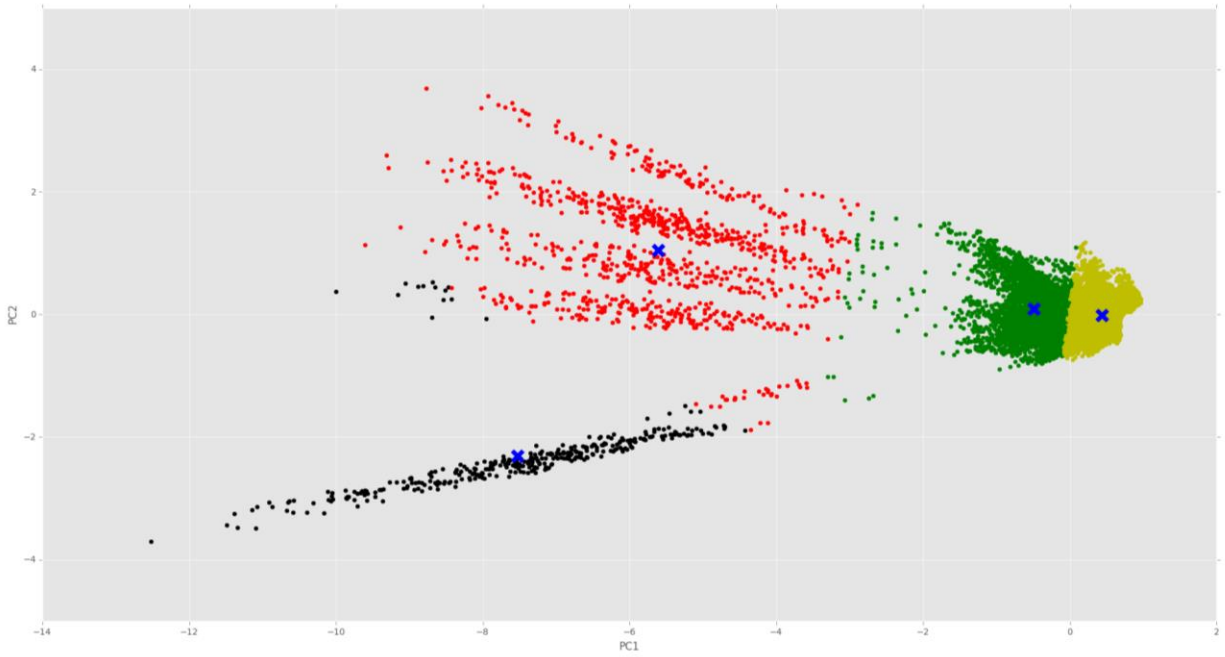


Figure (5.8) Flat Clustering of unlabeled Sound fault signatures at low speed tests on 2D scatter

The clustering criteria is still random and not satisfying the requirements. The samples were being classified based on different ranges of values of the first principle component (PC1). Vibration fault signatures have been fitted to the flat clustering method using K-Means algorithm. A two dimensional representation of pre-labeled data and flat-clustered data at low speed tests are shown in figure (5.9) and (5.10) respectively.

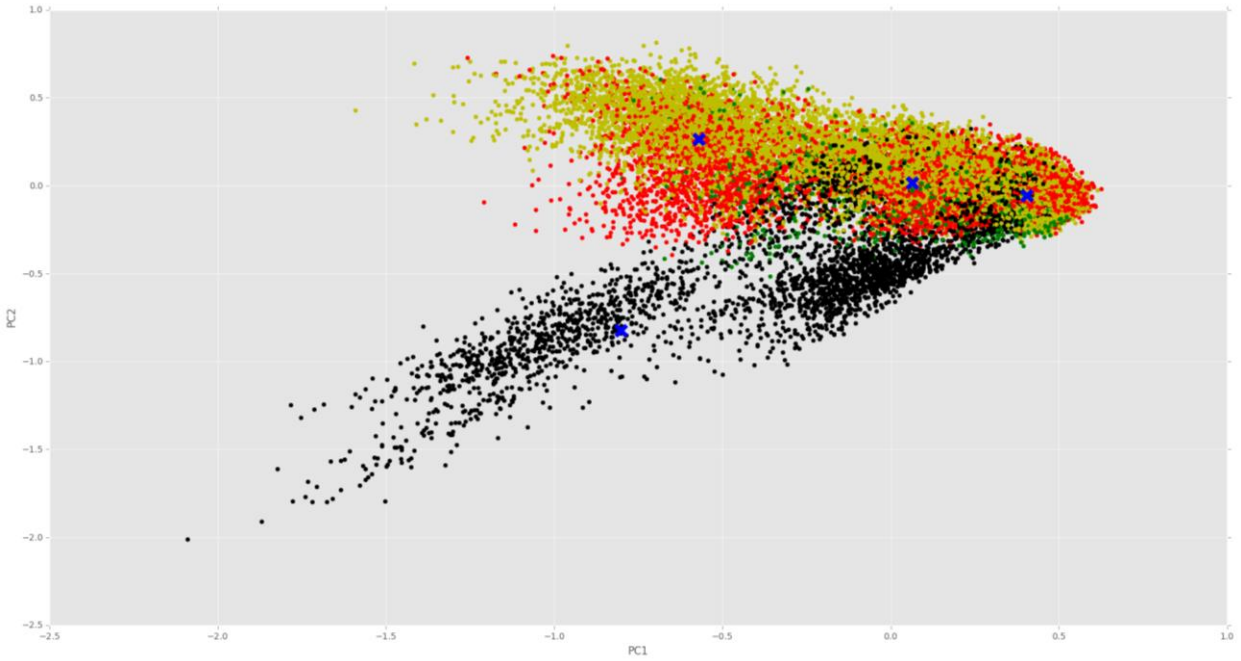


Figure (5.9) Labeled Vibration fault signatures at low speed tests on 2D scatter

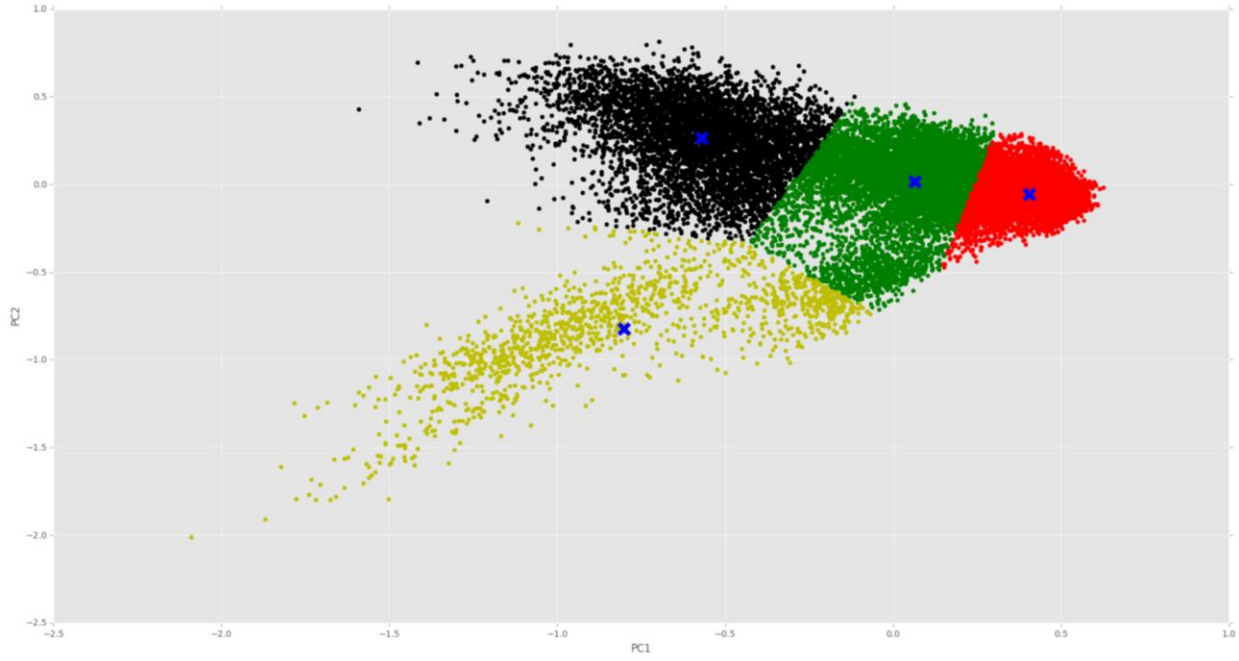


Figure (5.10) Flat Clustering of unlabeled Sound fault signatures at low speed tests on 2D scatter

Finally, the fault signatures of high speed tests for the vibration have been clustered. Figure (5.11) represents the supervised clustering while figure (5.12) captures the results of unlabeled flat clustering.

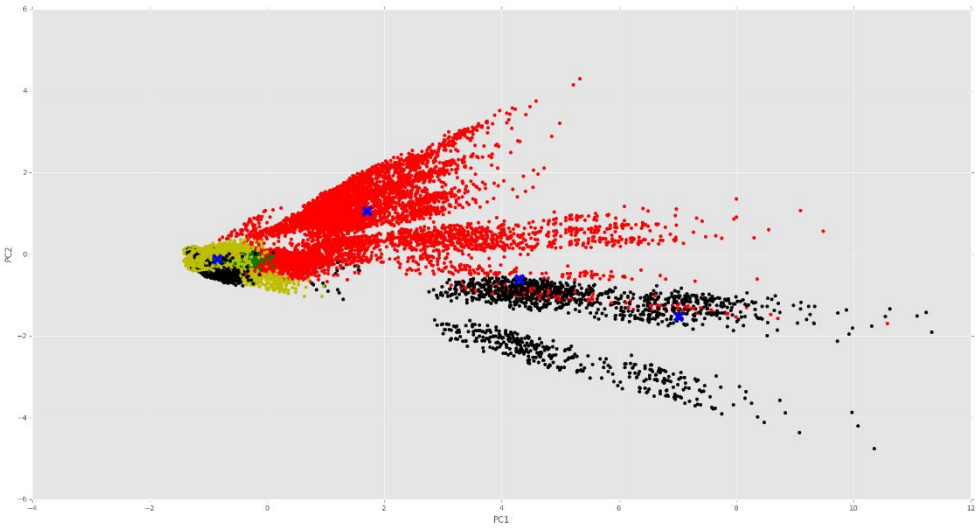


Figure (5.11) Labeled Vibration fault signatures at high speed tests on 2D scatter

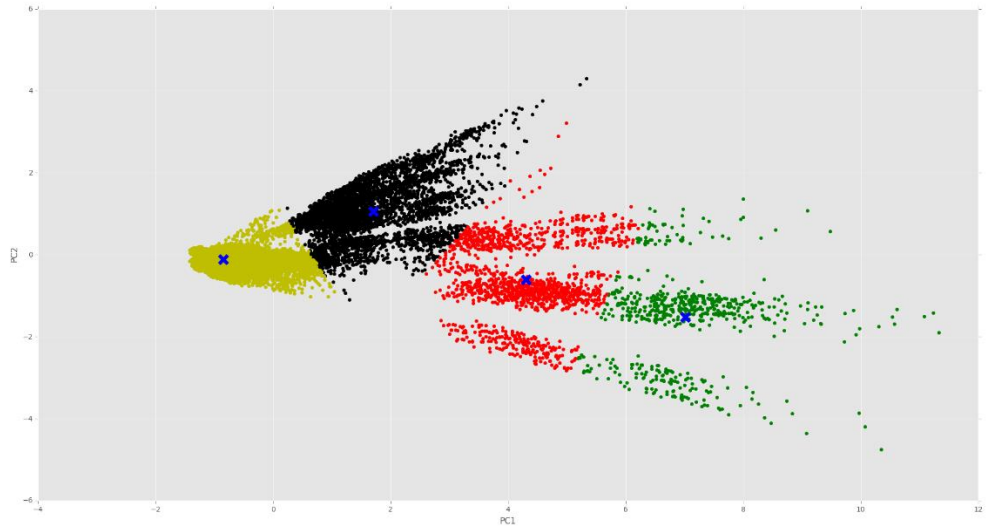


Figure (5.12) Flat Clustering of unlabeled Sound fault signatures at low speed tests on 2D scatter

It can be observed from the previous results of flat clustering that K-Means algorithm is not classifying the data in line with of the actual labels of the samples. One of the most common disadvantages of K-Means is that it creates crisp clusters. This is what is called "Divisive clustering Algorithm", which means that a sample cannot be shared within two clusters and one cluster group cannot contain samples from other clusters as well. It is noticed that the different types of faults of D&V data samples are having some common features, regardless of the significant differences. This makes the samples sharing the same areas, not geometrically apart, and even located exactly on the same coordinates.

5.4 Hierarchical Clustering

A different clustering method called Hierarchical clustering (a.k.a. Hierarchical Cluster Analysis HCA) has been applied to our data as a kind of unsupervised learning in order to compare its results with flat clustering and pick the best for the case. It seeks to build a hierarchy of clusters (See discussion and example in section 2.4.8). This method uses a centroid-based algorithm, known as Mean Shift algorithm. It involves shifting the kernel of the densest area iteratively to a higher density region until convergence. Mean Shift algorithm is actually calculation intensive which makes it take longer to process than other techniques. It also relies on having sufficient high density of data with clear gradient to locate the cluster centers. Furthermore, its boundaries are softer than K-Means algorithm and do not suffer from crisp clustering, which is called "Agglomerative Clustering Algorithm". It is expected to give more freedom for the samples to locate between different clusters. Similarly to previous cases, the Principle Component Analysis (PCA) was applied on the data before

it went through hierarchical clustering. In this method, everything was unsupervised, not only the data but the number of clusters as well.

Following the same order, the results start with sound fault signatures at low speed tests under hierarchical clustering as shown in two dimensions in figure (5.13) as well as a 3D representation in figure (5.14).

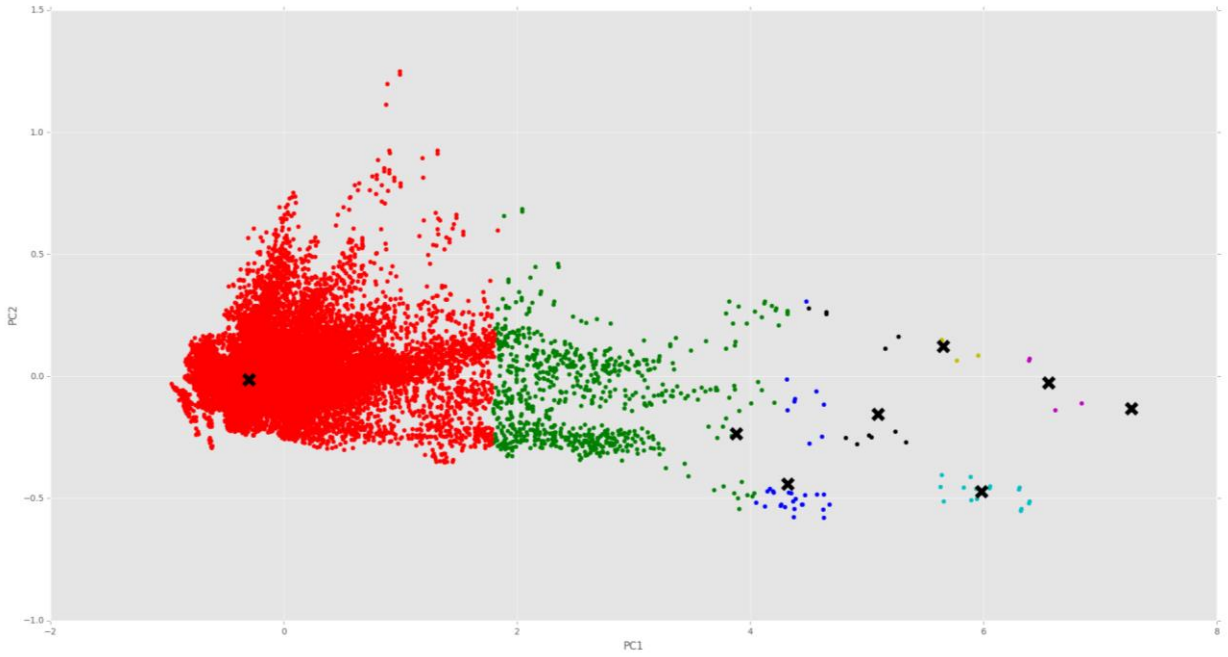


Figure (5.13) 2D Hierarchical Clustering of unlabeled Sound fault signatures at low speed tests

It is clear that this algorithm focuses on the high density areas to determine the number of clusters and which class each sample should belong to. The black "x" signs are the centroids of the clusters which means that the system split the dataset into 8 different clusters. It is good news that this method is more agglomerative (i.e. softer class boundaries as discussed in section 2.8.5) than K-Means algorithm but a higher number of clusters means that the system still cannot identify the most

important features yet. This is what makes neural networks do this job better as it is a divergent network that neglects all the undesirable information and looks for the unique features that best distinguish samples from one another.

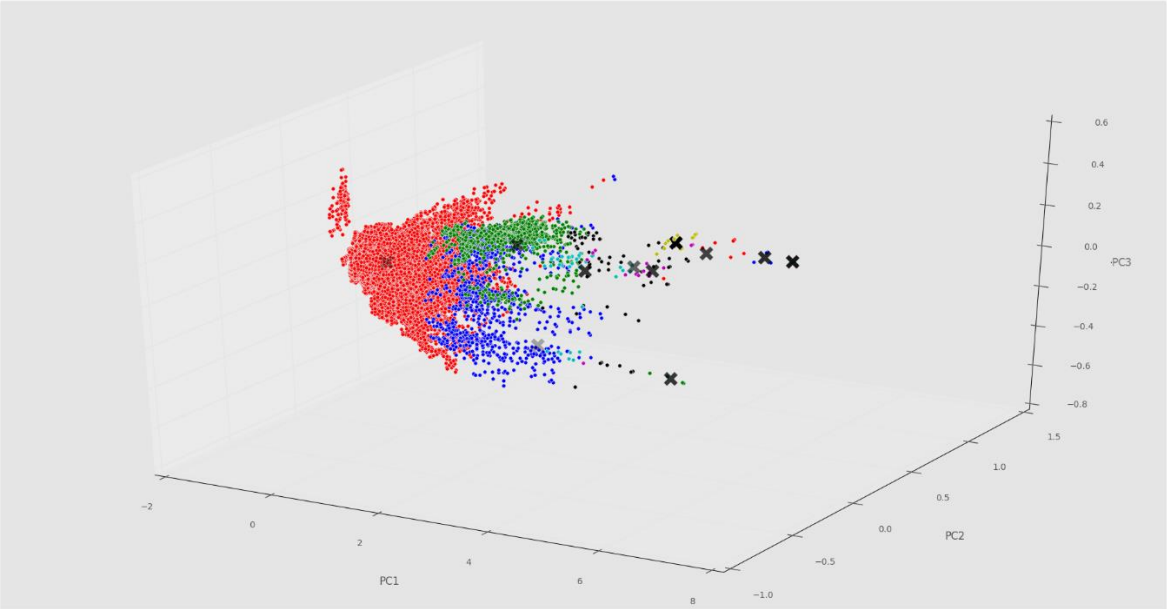


Figure (5.14) 3D Hierarchical Clustering of unlabeled Sound fault signatures at low speed tests

Let us try the fault signatures of the high speed test with and without feeding the system with the number of clusters with unlabeled data. Figure (5.15) and (5.16) show the two cases respectively. A big number of clusters (i.e. 16 clusters) was generated since there are many dense areas.

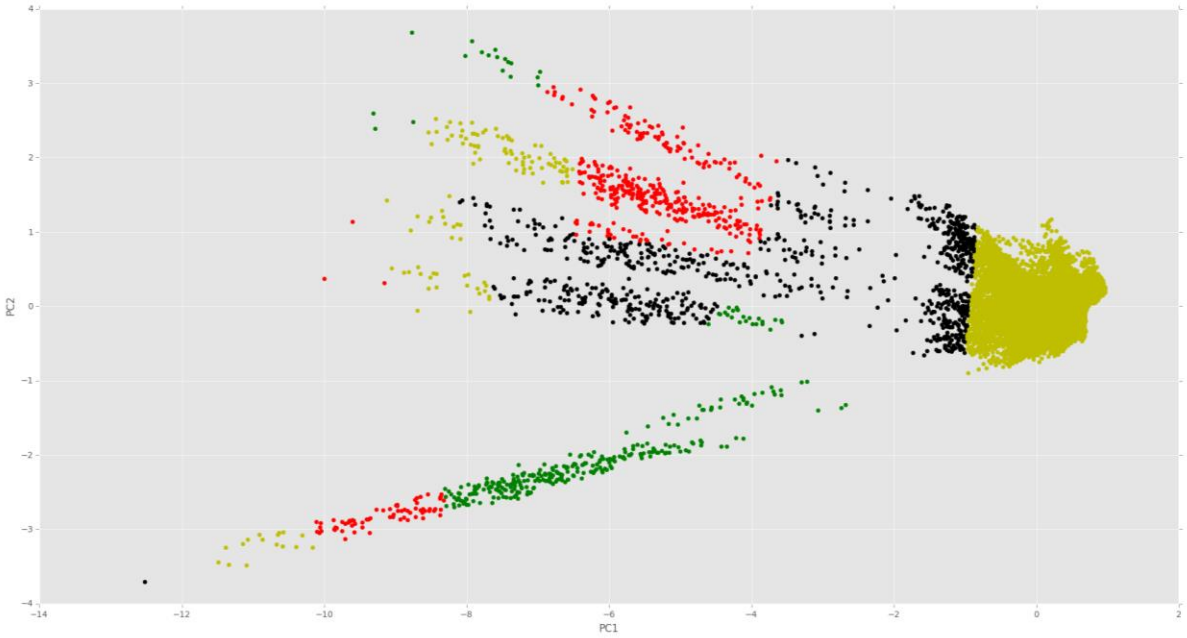


Figure (5.15) 2D Hierarchical Clustering of labeled Sound fault signatures at high speed tests

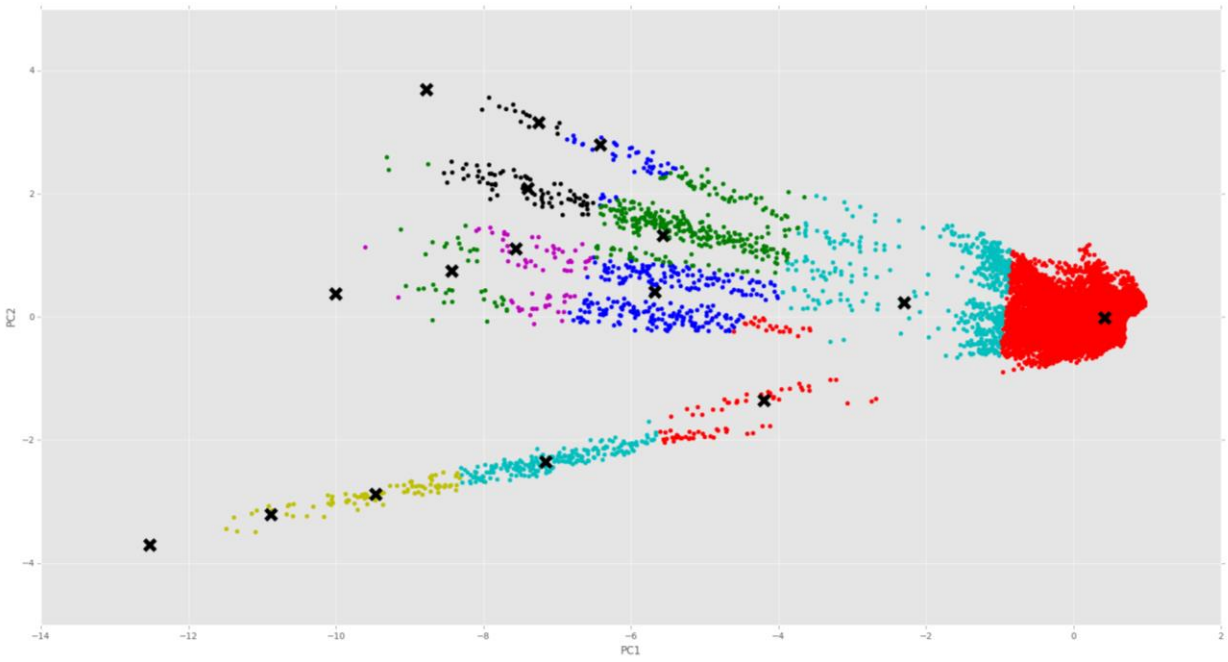


Figure (5.16) 2D Hierarchical Clustering of unlabeled Sound fault signatures at high speed tests

Vibration low and high speed tests clustering are shown in figures (5.17) and (5.18) respectively. A 3D representation of vibration at high speed test is also provided in figure (5.19) to show more details.

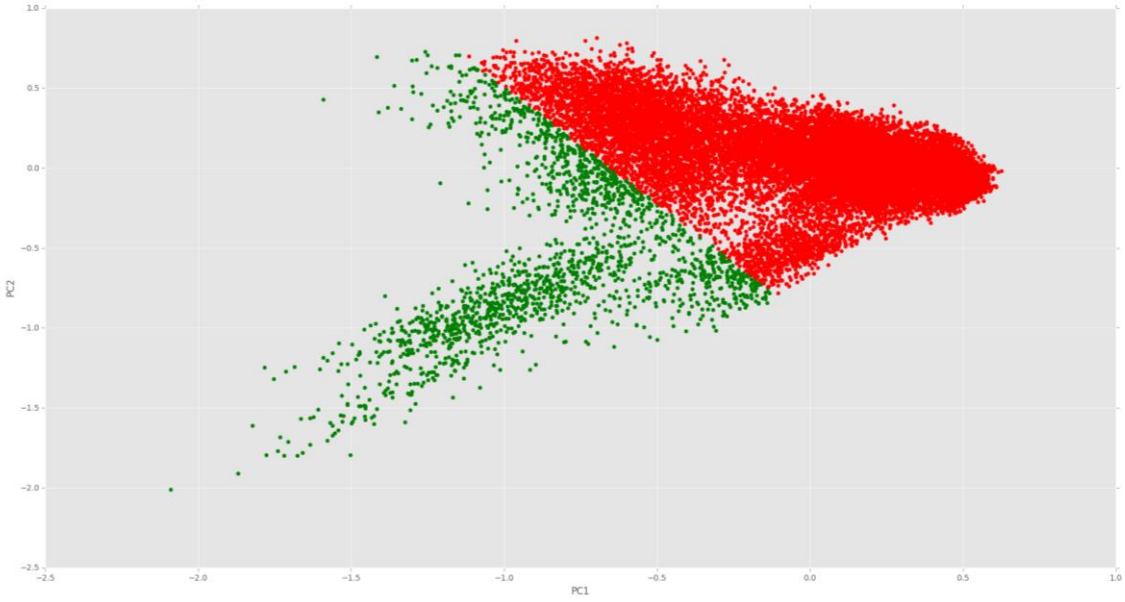


Figure (5.17) 2D Hierarchical Clustering of unlabeled Vibration fault signatures at low speed tests

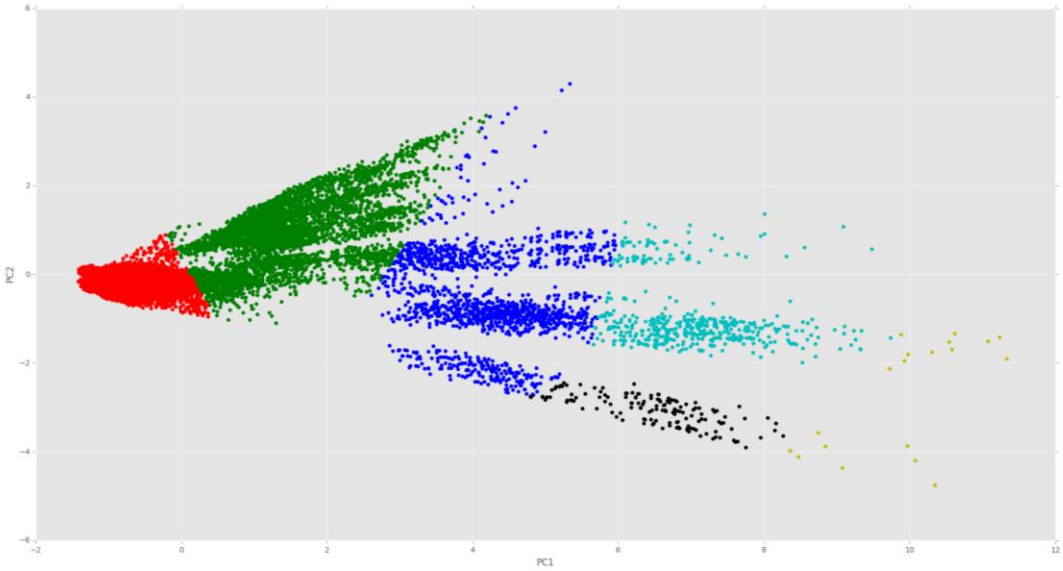


Figure (5.18) 2D Hierarchical Clustering of unlabeled Vibration fault signatures at high speed tests

Tests at 4200 rpm are giving better results than low speed as usual, that is because higher frequencies are richer with information rather than those at 1500 rpm. But compared to the actual scatter in figure (5.11), the classification of high speed tests samples is still missing a lot of samples from their right clusters.

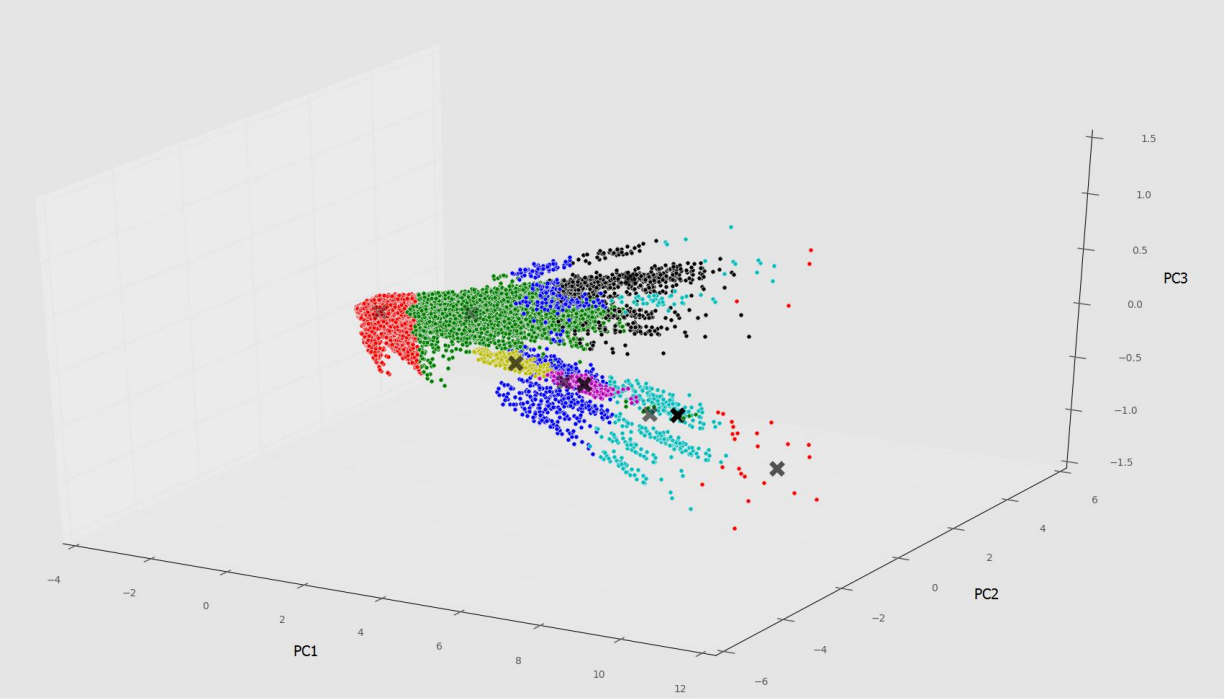


Figure (5.19) 3D Hierarchical Clustering of unlabeled Vibration fault signatures at high speed tests

An objective of this research was to determine if sound or vibration fault signatures would give higher performance, as well as low and high speed tests. Sound signals have shown to be slightly better than vibration signals. However, high speed tests greatly improve the performance in comparison to low speed tests. One way of improving the performance of the clustering algorithms, might be by using all the 64-dimensions that are available for each of the samples containing the sound, vibration under low speed and high speed tests. Implementing PCA over a 64-dimensional sample will result in new 2D representation, which is a weighted combination of 64 components reduced to 2 components (e.g.

PC1 and PC2). It is expected that some improvement would result. The results of labeled data (e.g. actual), unlabeled flat clustering (e.g. predicted) using K-means, and unlabeled hierarchical clustering using Meanshift are represented in figure (5.20), (5.21) and (5.22) respectively.

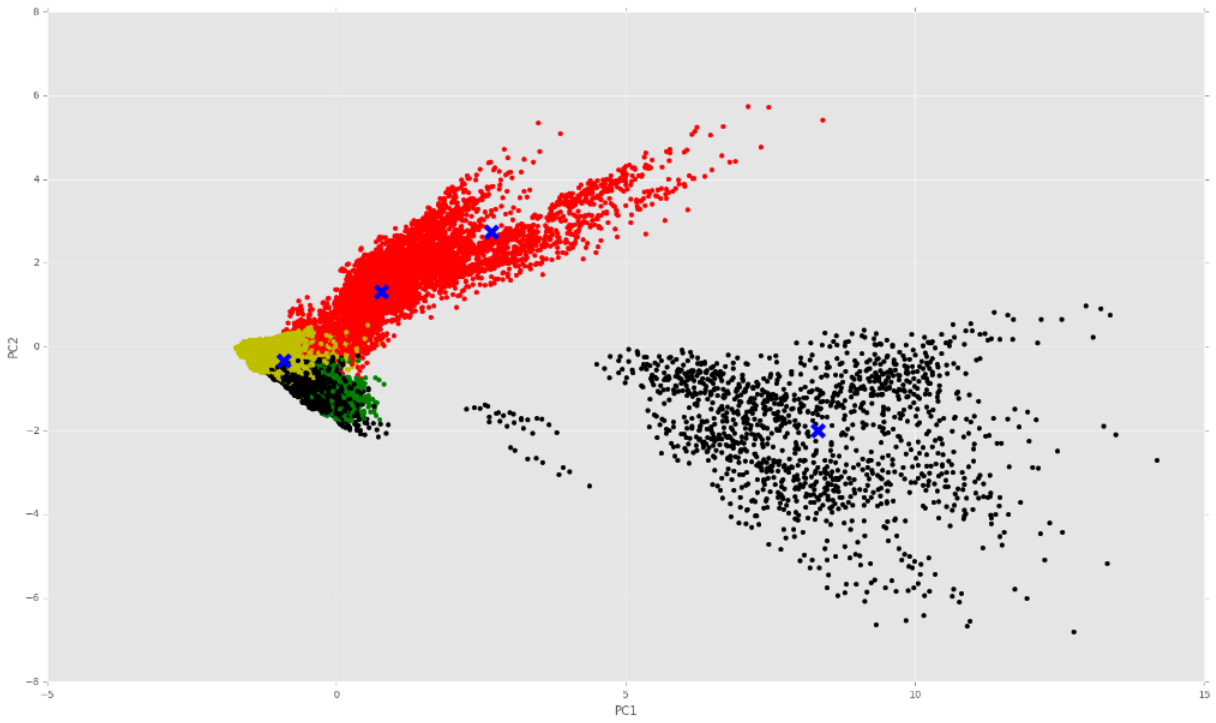


Figure (5.20) 2D Actual (labeled) clustering of all fault signatures

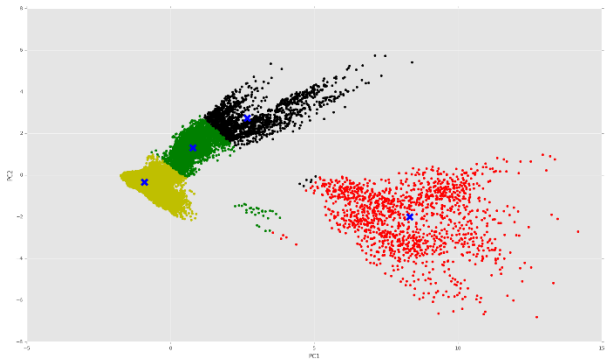


Figure (5.21) 2D flat clustering of all fault signatures

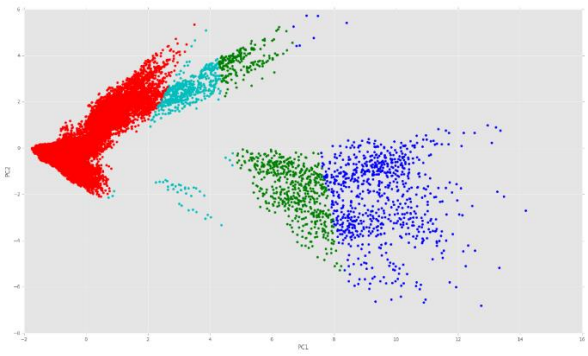


Figure (5.22) 2D Hier. Clust. of all fault signatures

Figure (5.23) and (5.24) represent the results of the same experiment but dealing with 3 principle components and plotting them on a 3D scatter.

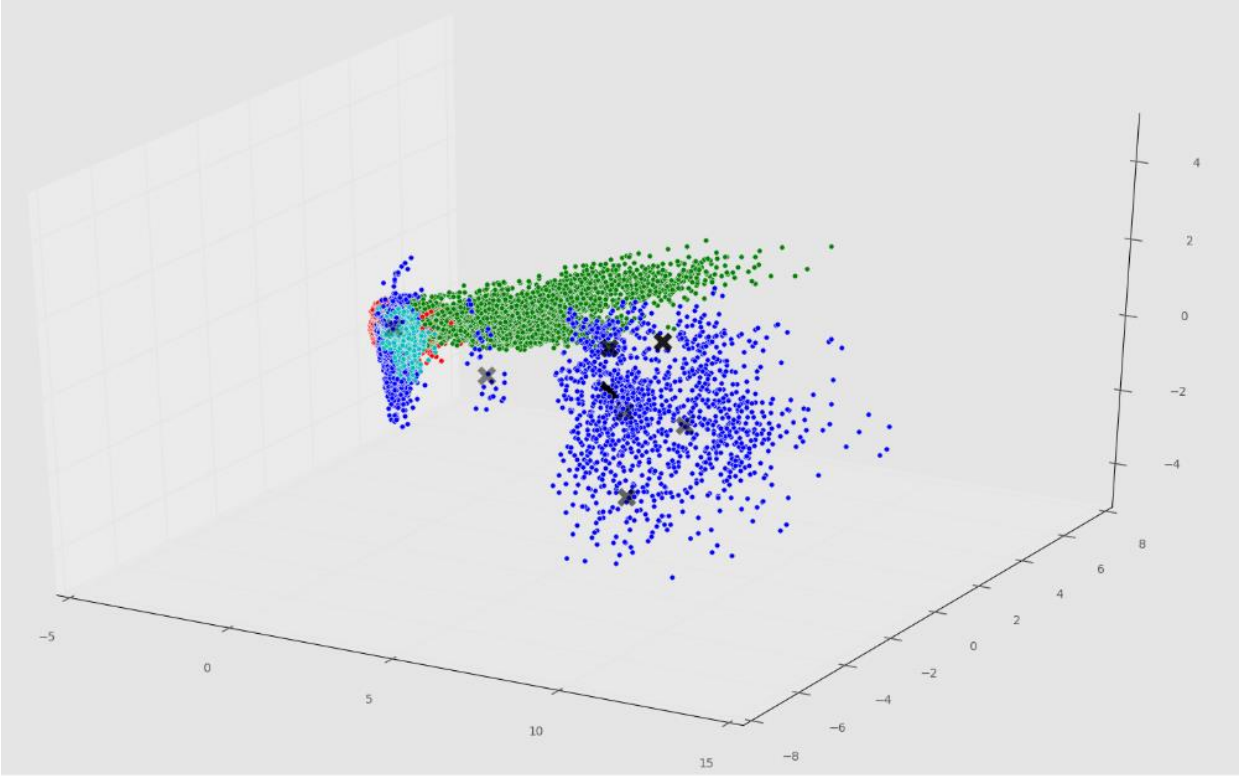


Figure (5.23) 3D flat clustering of all fault signatures

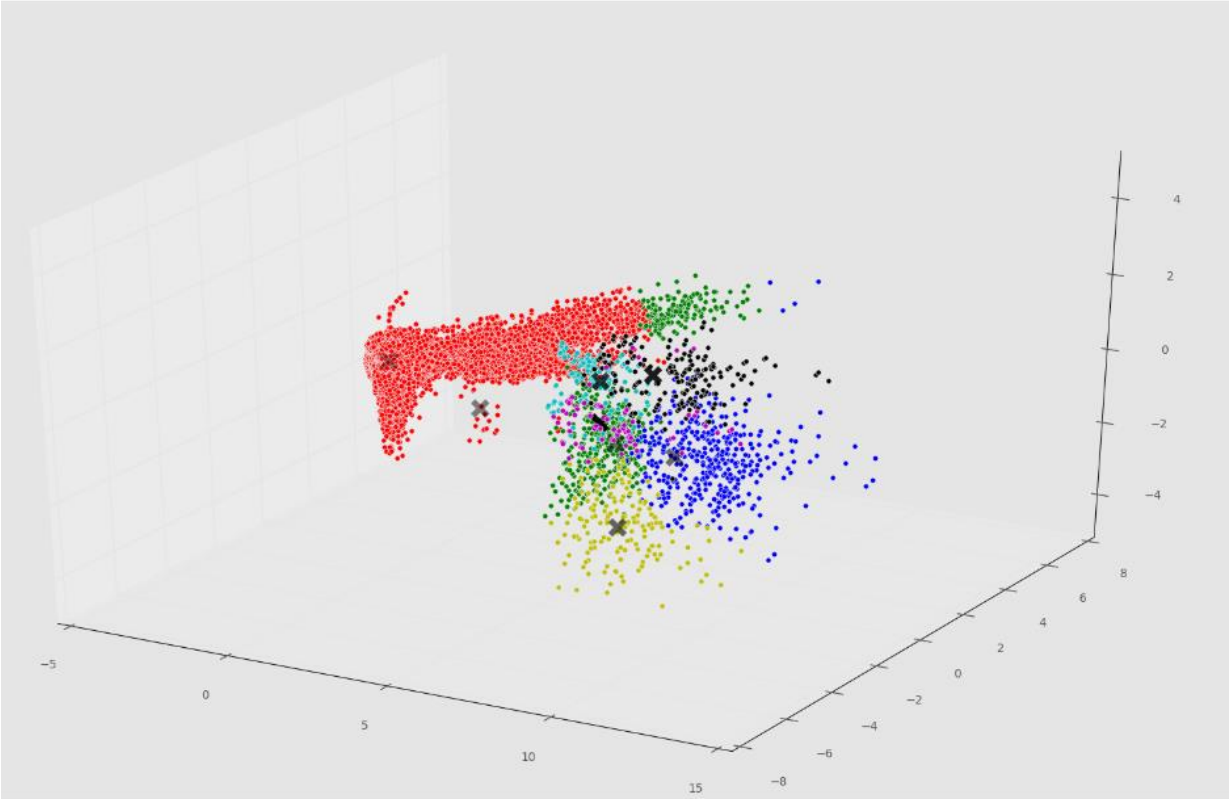


Figure (5.24) 3D Hierarchical Clustering of all fault signatures

The figures show an improvement in getting more realistic groups. The black points in the first figure are almost grouped the same as the ones in red in the second figure. Also the red group represents both green and black points as one group, but this time they have similarly close boundaries. However, some of the black points that are sharing areas with yellows and greens, were all predicted as one group. On the other hand, hierarchical clustering showed unrealistic results. Figures (5.23) and (5.24) show the best results that could be obtained using principle component analysis (PCA). Although data visualization and dimensionality reduction are important for observations and computation time, but the performance must have the priority at the end, otherwise visualization is useless.

5.5 Numerical Results

For this reason, clustering algorithms have been implemented over the whole dataset, with all its 26,983 64-dimensional samples without PCA. It will not be possible to visualize the results of this experiment since it is dealing with 64d points. However, a confusion matrix can clearly show the numerical results in details. Figure (5.25) shows the confusion matrix of the final results.

Predicted Value	Healthy	Armature Fault	Brush/Comm. Fault	Planetary Gear Fault	Number of Training Samples
Healthy	**80%**	**4%**	**9%**	**7%**	**14520**
Armature Fault	**26%**	**54%**	**11%**	**19%**	**7139**
Brush/Comm. Fault	**0**	**15%**	**71%**	**14%**	**4114**
Planetary Gear Fault	**27%**	**20%**	**18%**	**35%**	**1210**
Avg Detection Rate	**60%**				**26983**

Actual Value

Figure (5.25) Confusion Matrix of the results of all 64-dimensional samples

The average detection rate that has been achieved from the unsupervised learning model was 60%. It is concluded that in the common unsupervised learning applications, huge amounts of data is needed

for obtaining high accuracies. However, the results are never perfect, but using big data is the key for good generalisation. The evolution of the unsupervised learning results are shown in table (5.1) as well as the computation time for each.

	Flat Clustering		Hierarchical Clustering	
	Average Detection Rate	Process Time	Average Detection Rate	Process Time
32d + PCA (2d)	13%	~ 2 min	25%	~ 10 min
64d + PCA (3d)	18%	~ 6 min	38%	~ 15 min
High Speed Tests (32d)	25%	~ 7 min	46%	~18 min
All Fault Signatures (64d)	38%	~ 10 min	60%	~ 20 min

Table (5.1) Evolution of unsupervised learning results

5.6 Summary

An unsupervised learning algorithm might be able to discriminate different types of faults apart, but it is never capable to state which type of fault each of them belongs to. This is because it has no labels to guide or give any information about the class. However, a really helpful information is determining the number of output classes manually. This would avoid at least suggesting more classes than the actual ones. It is the researcher's responsibility to label the classes. In this research, an unsupervised learning model has been designed to take on only samples with absolutely no labels. Two clustering algorithms (e.g. K-Means and Mean-shift) have implemented to find similarities between samples. Every sample has been then grouped with its class neighbors. The best detection rate that has been achieved from the unsupervised learning model was 60%. The amount of data highly affects the performance of an unsupervised learning model as it needs a lot of examples for the system to make

sure its mapping are going in the right direction. Even the most popular applications the results are never perfect, but using big data is the key for a good mapping from the samples and thus good generalization.

Chapter 6

Conclusions and Future Work

This chapter provides a summary of the research, contributions, algorithms and results. Also, recommendations for future work are suggested.

6.1 Summary of the research

This thesis proposes novel methodologies to deal with industrial or traditional data for fault detection and diagnosis. A fault detection and identification algorithm has been developed to test electric motors. The development and results are based on real dataset obtained over several past years by D&V Electronics. This research was not focused on fault diagnosis algorithms using common concept, but relied on processed data in the form of fault signatures as proposed by M. Ismail [2].

In chapter 1, an overview on machine learning is provided. The implementation of machine learning techniques over industrial data was reviewed. Artificial intelligence (AI) is a wide field

accommodating all smart computer systems that mimic the duties of a human brain. Machine learning represents Artificial Intelligence (AI) when the system is learning from historical data. It is divided into 3 main types of learning: supervised learning, semi-supervised learning and unsupervised learning.

Chapter 2 discussed in details the theories as well as the different methods and algorithms for each of the 3 above mentioned learning types. The difference between learning with a teacher or without a teacher has been described. A teacher is basically the label of a sample. When labels are available, it is easy for the system to develop the relation between samples and their label. Learning without a teacher is relatively difficult. It requires mapping all data samples and coming up with general rules that works for all samples. This is what is called the generalization step, which is the most difficult and uncertain process that the learning model performs during the training process. Different learning algorithms that were suggested in this research have been discussed in details such as support vector machines (SVM), label propagation algorithm, flat and hierarchical clustering algorithms and neural networks. Deep networks and were also discussed in this chapter, as well as the conditions and requirements to apply them.

A supervised learning model is proposed in chapter 3 and applied to data provided by D&V Electronics. During the training process, all samples and labels that were allowed to enter the training path were fed to the machine learning algorithm. Two main algorithms have been built, tested and compared. The first algorithm was the artificial neural networks (ANN). It was composed of one input layer, multiple hidden layers that were adjusted depending on the amount and dimensions of the dataset and finally an output layer. The detection rate of the supervised model in predicting the

type of fault of samples was outstanding. Support Vector Machines were also designed to fit to the dataset, and their performance was comparable to the ANN.

Both artificial neural networks (ANN) and support vector machines (SVM) performed well. They actually reached high detection rates with relatively small numbers of training samples. SVM had a detection rate of 98% with only 200 training samples while ANN reached this value at 400 samples. However, ANN achieved the 100% detection rate at only 800 samples, while SVM achieved it with 1000 samples. The results indicated that with fewer labeled samples, SVM would work better than ANN. A detection rate of 100% is required with the least possible number of samples in such an application, ANN would be the better choice. The supervised learning model was able to diagnose new untypical fault types that the system has never been trained on. This is a very important feature that avoids false positives.

Chapter 4 represented a semi-supervised learning model. The challenge was to extract the features from a set of few labeled samples, and assign their labels to similar samples having the same features. Label propagation algorithm was in used in this task. It is a graph-based training algorithm that maps few labels from labeled motor samples to a relatively huge amount of unlabeled samples. The algorithm tries at all times to find similar communities. It starts with the few labels by assigning each label to its closest node in its community. The dataset begins the challenge with a big number of small communities, and ends up with a number of communities that is exactly equal to the number of possible classes (e.g. fault types). A smart technique called Active Learning technique was really helpful and effective in improving the performance of this model. This technique starts labeling a very small number of samples that is determined by the user. It does not select samples randomly, but chooses them based on the most uncertain samples to label first to avoid error accumulation. The

number of uncertain samples also could be determined manually. In this research, it started by 10 labels and ended with 30 labels with a step of 5 labels in each iteration. In comparison with supervised learning with the same number of labels (i.e. 50 labels), the detection rate was 93% while in semi-supervised learning it is 84%. This is because support vector machines and neural networks receive these labels as certain information which they are safely allowed to create rules from. However, in the semi-supervised model, the decisions are taken mainly from uncertain information and the labels are just for guidance to start off the propagation wave. Thus, the model is less capable to generalize. The detection rate remained 83% from 45 labels to 30 labels and 370 unlabeled samples and decreased to 82% at 25 labels. Then, it drops down to 30% with only 10 labels.

Unsupervised learning was the real challenge in this research. A model has been designed to take on only samples with absolutely no labels. The learning algorithm extracted samples features and performed a mapping to find out si milarities between samples. An unsupervised learning algorithm has no labels to guide, so it might be able to discriminate different types of faults apart, but it never states which type of fault they belong to. The detection rate of an unsupervised learning model is highly affected by the amount of data. The highest average detection rate that has been achieved from the unsupervised learning model applied to D&V data was 60%.

6.2 Outcomes and Concluding remarks

The fault detection and identification system that has been developed in this research using machine learning techniques has achieved the following outcomes:

1. A supervised learning model with 100% average rate of detection and diagnosis using only 800 labeled training samples out of 26,983. This model is capable of diagnosing and detecting new untypical fault types.
2. A semi-supervised learning algorithm with 83% average detection rate with only 30 labels and 370 unlabeled samples.
3. An unsupervised learning model with 60% average detection rate. The main reason is the relatively small amount of data that was available.

6.3 Recommendations and Future Work

There is no direct answer to the question pertaining to the best learning algorithm. It always depends on the available resources, the type, size, amount of data, and the available amount of labeled data. If there is a sufficient amount of labeled samples, thus they should be used. Unsupervised learning is challenging because of lack of labels. Labeled instances are often difficult, expensive, and time consuming to obtain, as they require human intervention. Unlabeled data is relatively easy to collect, but there has been few ways to use them. Semi-supervised learning addresses this problem by using large amount of unlabeled data, together with the labeled data, to build better classifiers. Based on our results, Support Vector Machines (SVM) are recommended as they typically show good results in multiple applications.

Unsupervised learning and deep learning both have a common requirement, which is big data. Getting a big number of unlabeled samples definitely helps in better mapping and generalization. The typical software applications that are developed with unsupervised learning need hundreds of thousands of unlabeled samples for a high detection rate.

Appendix A

Figure (A.1) shows the architectural layout of a multilayer perceptron (MLP). The backpropagation algorithm will be applied to this example. It is a neural network composed of one input layer, one output layer and two hidden layers.

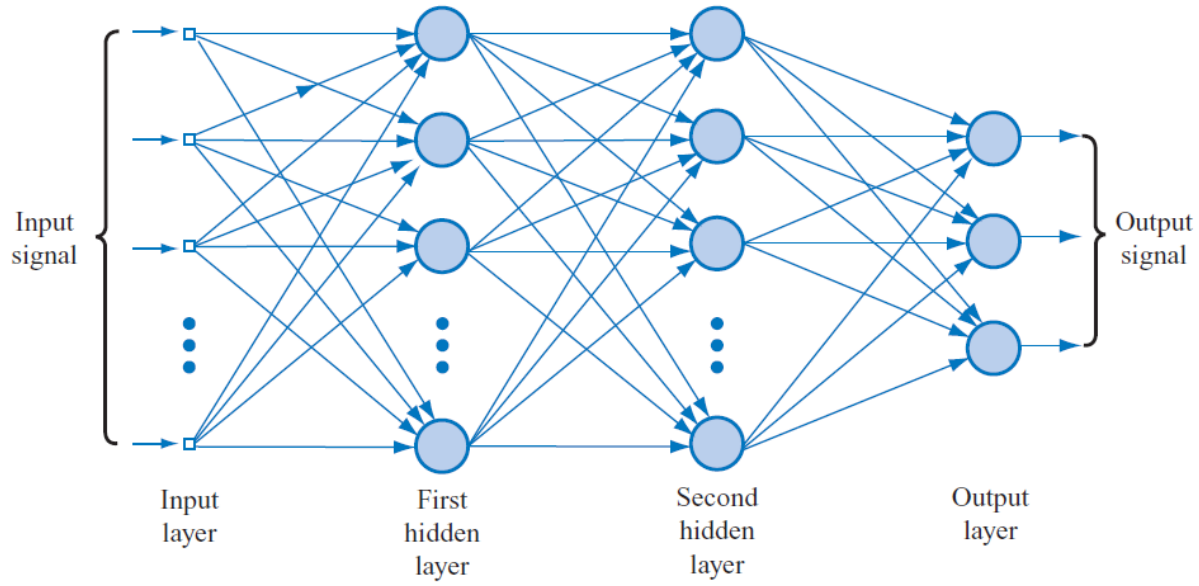


Figure (A.1) MLP [8]

The input layer contains a number of neurons (x_n) and the network should end up with one of three desired responses (d_n) in the output layer. Each unit is fully connected to all the units of the previous layer as mentioned before. Figure (A.2) represents the signal flow of the backpropagation algorithm corresponding this network. The upper part of the graph is the forward phase while the lower part is the backward phase during the learning process. The backward pass is actually referred to as a sensitivity graph for computation of the local gradients in the backpropagation algorithm. [Narenda and Parthasarathy, 1990] [9]

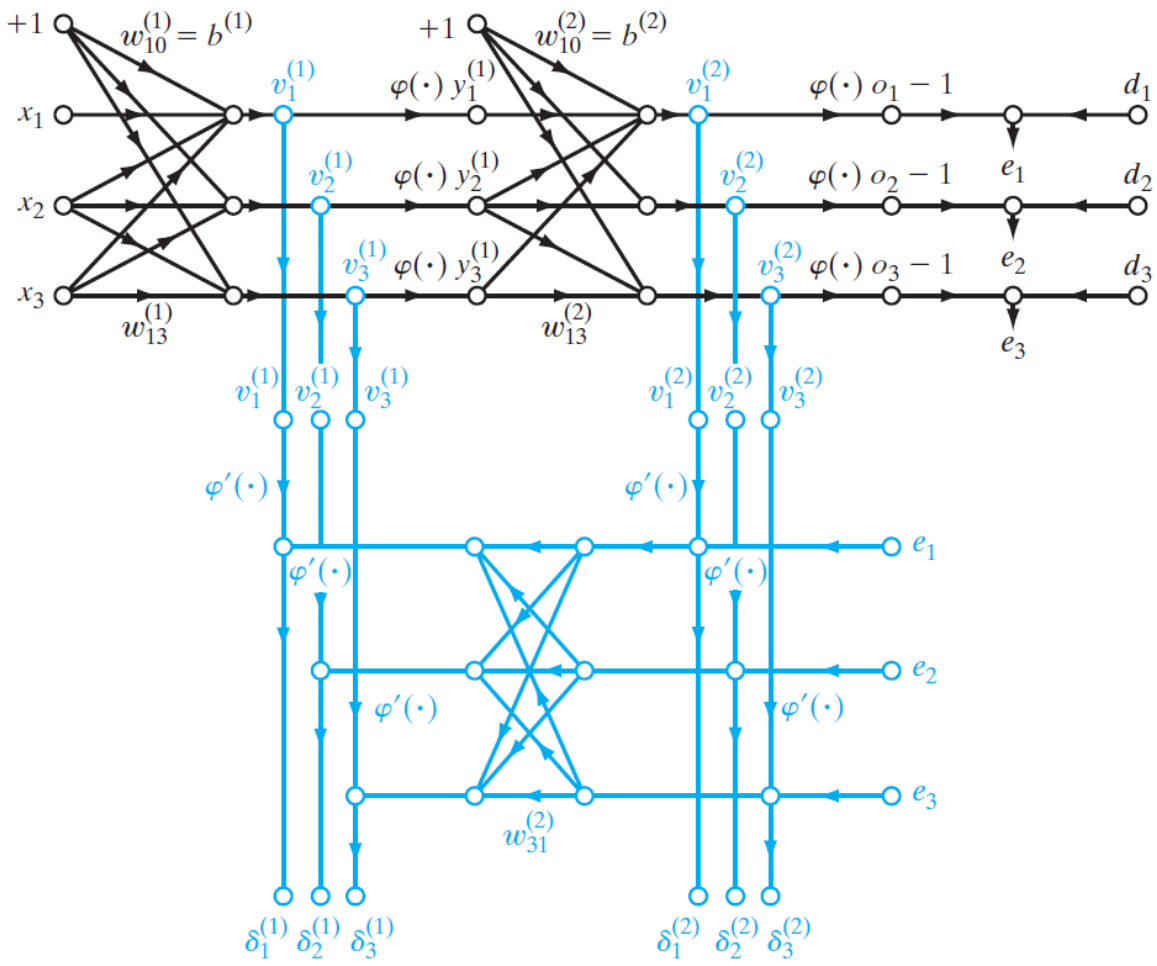


Figure (A.2) Signal Flow of Backpropagation algorithm [8]

The algorithm starts by the initialization step, where it is assumed that no prior information is provided. Both the synaptic weights and thresholds are picked from a uniform distribution. The mean of this distribution is zero and its variance is chosen to make the standard deviation of the induced local fields of the neurons. For the forward computation, the induced local fields and function signals of the network are computed by proceeding forward through each layer of the network. The induced local field $v_j^l(n)$ for neuron j in layer l is:

$$v_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{A.1}$$

$y_i^{(l-1)}$ is the output signal of neuron *i* in the previous layer *(l-1)* at iteration n while $w_{ji}^{l}(n)$ is the synaptic weight of neuron *j* in layer *l* fed from neuron *i* in layer *(l -1)*. On the way back (Backward computation), the local gradients (δ_s) of the network are computed using the following equation:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)) & \text{for neuron } j \text{ in output layer } L \\ \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) & \text{for neuron } j \text{ in hidden layer } l \end{cases} \tag{A.2}$$

Where *e* is the corresponding error signal and φ is the linear function and the prime on it means its differentiation with respect to the argument. The updated synaptic weights are calculated using the generalized delta rule:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\Delta w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n) y_i^{(l-1)}(n) \tag{A.3}$$

Where ɳ is the learning rate parameter and is the momentum constant.

Finally, new epochs of training examples are iterated to the network until the required stopping criteria is achieved. As the number of training iterations increases, the momentum and learning rate parameter decrease.

Appendix B

Experimental Datasets and Setup

All experimental data and measurements that were used for the testing of the FDD algorithm [2] were provided by D&V Electronics. A high production starter tester ST-118 shown in figure (B.1) was used to produce these data with the following specifications:

- Starter Current up to 1600 Amps
- Starter Voltage 12 & 24 Volts (32V optional)
- Starter Speed 0-20,000 RPM
- Starter Torque up to 50 Nm
- Solenoid Current up to 150 Amps
- Contact Voltage Drop 0-2.5 Volts
- Starter E_ciency 0-100%
- Starter Output Power up to 2.5 kW
- Starter Input Power up to 20 k
- Ripple Current 0-200 Amps tabs

More information about the tester can be found in [85]

Figure (B.1) D&V Starter Tester ST-118 [2]

The test profile of this tester model had 2 rotational speed profiles for the test as shown in fig (B.2).

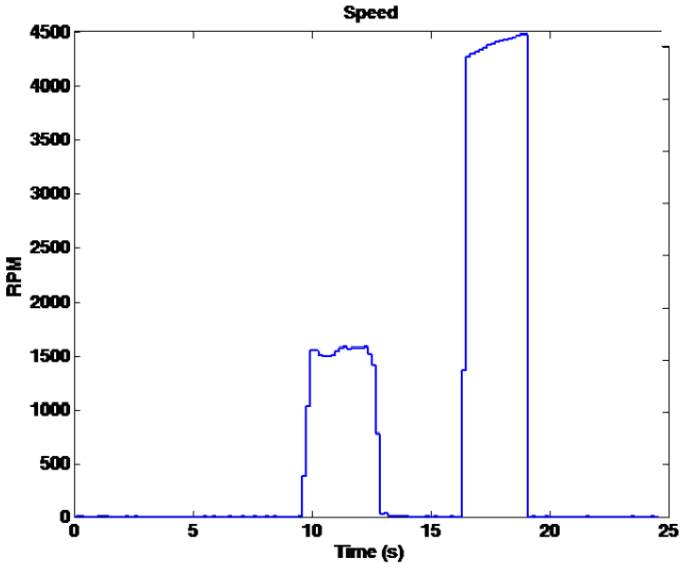


Figure (B.2) Low and high speed test profiles [2]

The low profile represents an average speed of 1500 rpm while the high profile represents an average speed of 4200 rpm. One microphone and one accelerometer were used to capture the sound and vibration measurements. Those sensors were located at the same cell where the starters and alternators as shown in fig (B.3).

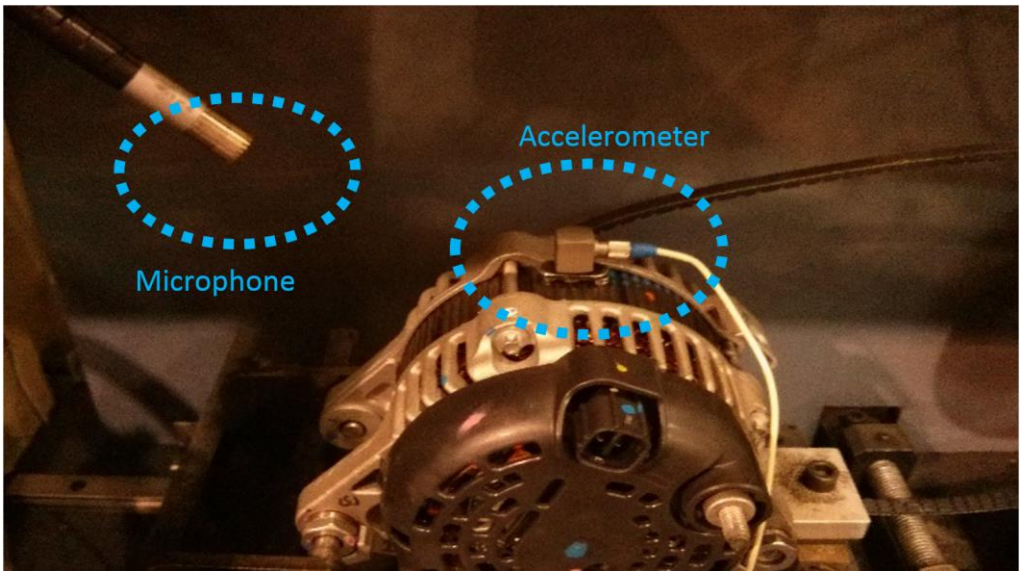


Figure (B.3) Sound & Vibration sensors locations [2]

The sound and vibration measurements were provided as raw signals recorded by these sensors. Fig (B.4) shows an example of these measurements.

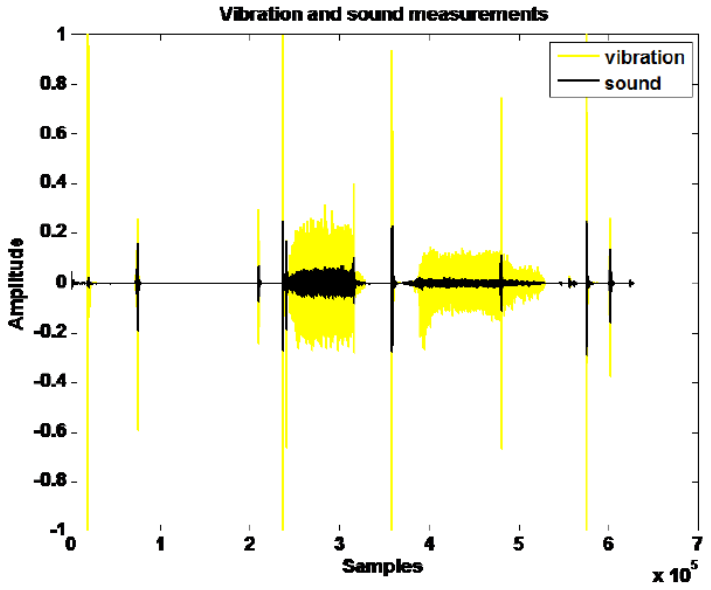


Figure (B.4) Sound & vibration measurements example [2]

These measurements were analyzed and processed using IEMSPCA algorithm that was discussed in Chapter 3. All the parameters that were used for the analysis are listed in Table (B.1).

Symbol	Parameter name	value
Background noise elimination feature		
τ	time window size	4096 samples
γ	spectrum size	4096 samples
A	Attenuation range	-60 dB
ω	frequency smoothing width	100 Hz
$\beta, \zeta, and\ \rho$	attack, release and hold times	0.1 % of total time of the signal
EMSPCA feature		
j	wavelet levels	4 levels
ψ	wavelet family	Daubechies, order 16
F_s	sampling frequency	44.1 Khz
p	PCA number of variables thresholded	1 variable
Classification feature		
C_{size}	numerical method step size	0.05
$C_{max\ step}$	numerical method maximum iterations	500
w_{FLD}	gain vector initialization method	Fischer linear discriminant
$C_{tolerence}$	convergence tolerence	0.01

Table (B.1) Parameters used for the analysis of sound and vibration measurements using IEMSPCA [2]

Appendix C

Fault Signature Classification using FDD

Different random effects may affect sound and vibration measurements in industrial environments, such as the humidity that might slowly affect acoustics. Thus, a dynamic classifier was used for the fault signatures to allow small and meandering bias in the manufacturing system to occur. Not all the healthy and faulty windows (i.e. sub-training set having signatures that correspond to a certain class) were renewed, only a portion of the training windows (e.g. half of the training set) were updated. Thus, half of the initial historical data that was labeled by an experienced technician was retained, while the rest were excluded and replaced dynamically. The bias was removed with careful continuous classifier updating as following:

- The classifier was initialized and trained with the labeled training dataset.
- The healthy signature with the highest score was selected as the new baseline.
- New measurements were obtained as well as their fault signatures.
- If a fault signature was classified as healthy by the classifier, one healthy signature in the updating half of the healthy window was replaced by the new signature in First Input First Output (FIFO) manner.
- If a fault signature were classified as faulty by the classifier, one faulty signature in the updating half of the fault window was replaced by the new signature in First Input First Output (FIFO) manner.
- All previous steps were repeated and new measurements were obtained.

The following flowchart in figure (C.1) summarizes all the previous steps.

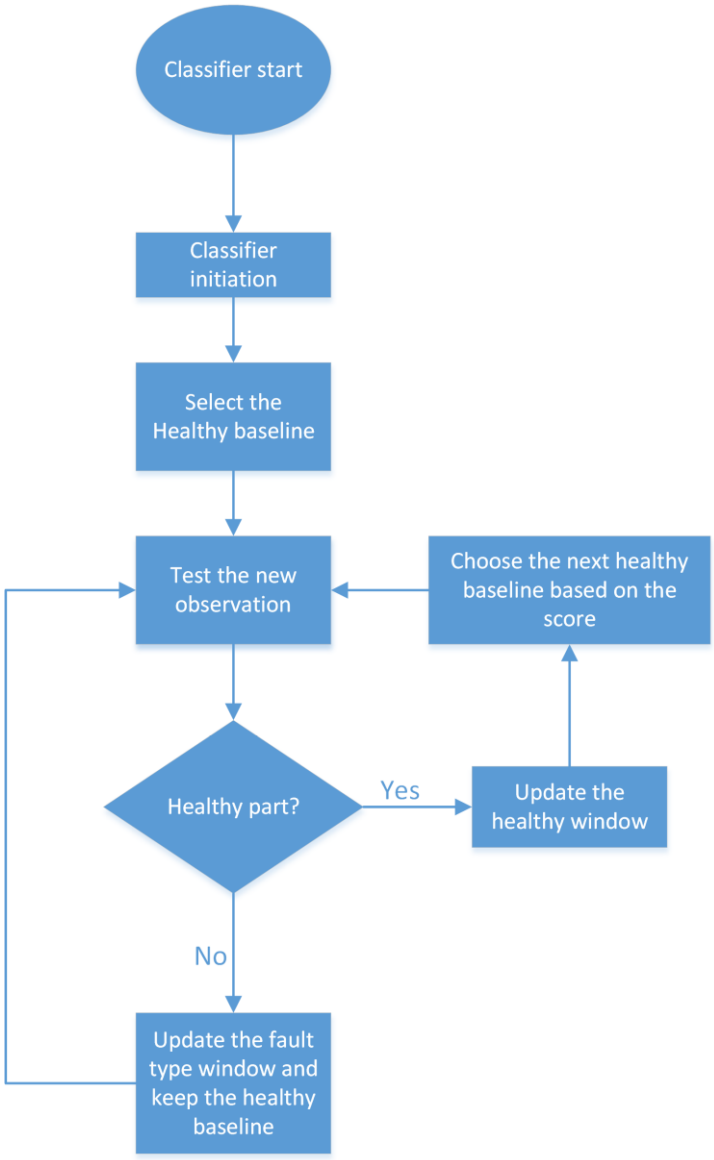


Figure (C.1) Classification process in FDD solution [2]

References

[1] Y. Benjio, J. Goodfellow and A. Courville, Deep Learning, Book in preparation for MIT Press, 2016.

[2] M. Ismail, Industrial Extended Multi-Scale Principle Components Analysis for fault detection and diagnosis of car alternators and starters, M.Sc. Thesis, McMaster University, 2015.

[3] S. Haykin, Neural Networks and learning Machines, Third Edition ed., New Jersey: Pearson, 2009.

[4] D. J. Mackay, Information Theory Inference and Learning Algorithms, Cambridge: Cambridge University Press, 2003.

[5] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th edition ed., Vols. ISBN: 978-1-59749-272-0, Elsevier, 2009.

[6] UofT, "Coursera," [Online]. Available: www.coursera.org/learn/nerural-networks.

[7] I. Velsiev, "A Deep Learning Tutorial: From Perceptrons to Deep Networks," 2010. [Online]. Available: https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks.

[8] S. Haykin, Neural Networks: A Comprehensive Foundation 2nd, vol. ISBN:0132733501, New Jersey, USA: Prentice Hall PTR Upper Saddle River, 1998.

[9] S. Haykin, Cognitive Dynamic Systems, Cambridge University Press, 2012.

[10] M. A. Nielson, Neural Networks and Deep Learning, Determination Press, 2015.

[11] D. D. Team, "Deeplearning4j: Open-source distributed deep learning for the JVM," Apache Software Foundation License 2.0, [Online]. Available: http://deeplearning4j.org.

[12] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, Vols. Vol.2, No.1, no. DOI: 10.1561/2200000006, p. 1–127, 2009.

[13] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, no. arXiv:1404.7828v4 [cs.NE], pp. 85-117, 2015.

[14] L. Gomes, "Machine-Learning Maestro Michael Jordan on the Delusions of Big Data and Other Huge Engineering Efforts," *IEEE Spectrum*, 2014.

[15] R. Dechter, "Learning while searching in constraint-satisfaction problems," in *AAAI-86 Proceedings*, Los Angelos, 1986.

[16] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Now publishers*, no. MSR-TR-2014-21, 2014.

[17] R. Socher, D. Chen, C. Manning and A. Ng, "Reasoning with neural tensor networks for knowledge base completion.," in *Neural Information Processing Systems (NIPS)*, Stanford, 2013.

[18] G. E. Hinton, A. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors.," *arXiv*, vol. 1, no. 1207.0580, 2012.

[19] G. E. Hinton, "Deep Belief Networks," *Scholarpedia*, vol. 4(5), no. 5947, 2009.

[20] P. e. al., "Scikit-learn: Machine learning in Python," no. JMLR 12, pp. 2825-2830, 2011.

[21] D. Albanese, G. Merler, S. Jurman and R. Visintainer, "MLPy: high-performance Python package for perdictive modeling," *NIPS, MLOSS workshop*, 2008.

[22] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A survey.," *ACM Comput*, Vols. 41,3, no. 10.1145/1541880.1541882, p. 58, 2009.

[23] C. Manikopoulos and S. Papavassiliou, "Network intrusion and fault detection: A statistical anomaly approach," *IEEE Comm. Mag. 40*, 2002.

[24] V. Chandola , A. Banjeree and V. Kumar, "Anomaly Detection for Discrete Sequences: A survery," *IEEE Transactions on knowledge and data engineering*, vol. 24, no. 5, 2012.

[25] M. Markou and Singh S., "Novelty Detection: A review-part 2: Neural network based approaches.," *Sig Proc. 83*, vol. 12, pp. 2499-2521, 2003.

[26] G. E. Hinton, S. Osindero and Y.-W. Teh, A fast learning algorithm for deep belief nets, Massachussets: MIT, 2006.

[27] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou and P. Barbano, "Toward automatic phenotyping of developing embryos from videos," *IEEE Transactions on Image Processing*, vol. 14(9), pp. 1360-1371.

[28] "Deep Learning," 2016. [Online]. Available: www.deeplearning.net.

[29] Y. Bengio, O. Delalleau and Simard C., "Decision trees do not Generalize to New Variations," *Comptuational Intelligence*, vol. 26, no. 4, pp. 449-467, 2010.

[30] I. StstSoft, Tulsa, OK: StatSoft, Electronic Statistics Textbook, 2013.

[31] B. Bergeron, Essentials of CRM: A guide to customer relationship management, NY: Wiley, 2002.

[32] X. Zhu and A. B. Goldberg, "Introduction to Semi-Supervised learning.," no. 10.2200/S00196ED1V01Y200906AIM006, 2009.

[33] A. Azran, "The rendezvous algorithm: multiclass semi-supervised learning with Markov random walks," *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007),* no. 10.1145/12734, p. 49–56, 2007.

[34] D. Sculley, "Results from a Semi-supervised Feature Learning Competition," *Google Pittsburgh,* 2011.

[35] M.-F. Balcan and A. Blum, "A PAC-cycle for learning from labeled and unlabeled data," *COLT,* no. 10.1145/1273496, 2005.

[36] Q. V. Le, A. Ranzato, Monga Rajat, M. Devin, K. Chen , G. S. Corrado, J. Dean and A. Y. Ng, "Building High-Level Features Using Large Scale Unsupervised Learning," in *Proceedings of the 29th International Conference on Machine Learning,* 2012.

[37] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI in large scale kernel machines," *MIT Press,* 2007.

[38] J. Sanz, R. Perera and C. Hueberta, "Fault Diagnosis of Rotating machinery based on auto-associative neural networks and wavelet transforms," *ELSEVIER, Journal of Sound and Vibration ,* vol. 302, pp. 981-999, 2007.

[39] M. Berry, A. J. and G. S. Linoff, mastering data mining, NY: Wiley, 2000.

[40] G. E. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," vol. 1, no. UTML-TR 2010-003, 2010.

[41] M. Markou and S. Singh, "Novelty detection: A review-part 1," *Statistical approaches Sig. Proc. 83,* vol. 12, pp. 2481-2497, 2003.

[42] H. Lee, Grosse Roger, R. Ranganath and A. Y. Ng, "Unsupervised Learning of Hierarchical representations with convolutional deep belief networks," vol. 54, no. 10.1145/2001269.2001295, p. 10, 2011.

[43] D. Erhan, A. Courveille, Y. Bengio and P. Vincent, "Why Does Unsupervised Pre-training Help Deep Learning?," *Proceedings of AISTATS 2010,* pp. 201-208, 2010.

[44] J. Zupan, Clustering of large data sets, NY: Research Studies Press, 1982.

[45] Y. Bengio, J. Louradour, R. Collobert and J. Westen, Curriculum Learning, Universite de Montreal, 1330, 2009.

[46] A. Ng, "CS229 Lecture Notes," 2015.

[47] A. Smola and S. Vishwanathan, Introduction to Machine Learning, NY: Cambridge University Press, 2008.

[48] S.-l. developers. [Online]. Available: http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html.

[49] G. Attardi, F. Dell'Orletta, M. Simi and J. Turian, "Accurate Dependency Parsing with a Stacked Multilayer Perceptron," in *Proceedings of Evilita 2009*, 2009.

[50] A. Coates and A. Y. Ng, "Learning Feature Representations with K-Means. Orignially published in Neural Networks: Tricks of the Trade, 2nd edn, Springer LNCS 7700," *Springer LNCS 7700*, vol. 2, 2012.

[51] A. Agarwal and Triggs B., "Hyperfeatures: Multilevel local coding for visual recognition," *9th European Conference on Computer Vision*, vol. 1, pp. 30-43, 2006.

[52] Mathworks. [Online]. Available: http://www.mathworks.com/help/fuzzy/examples/fuzzy-c-means-clustering.html.

[53] H. Larochelle, "Etude de techniques d'apprentissage non-supervise pour l'amelioration de l'entralinement supervise de modeles connexionnistes," PhD Thesis, 2009.

[54] R. Nock and F. Nielson, "On Weighting Clustering," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 28 (8), pp. 1-13, 2006.

[55] J. K. Aggarwal and M. S. Ryoo, "Human activity analysis: A review," *ACM Comput. Surveys*, vol. 43, no. 3, p. 16, 2011.

[56] J. C. Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms," no. ISBN 0-306-40671-3, 1981.

[57] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[58] H. Geoffry and R. Salakhutdinov, "reducing the Dimensionality of Data with Neural Networks," *Science 313*, pp. 504-507, 2006.

[59] J. A. Hartigan and M. A. Wong, "Algorithm 316. A K-Means Clustering algorithm," *Applied Statistics*, vol. 28, p. 100.

[60] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag and T. Moriarty, "A Modified Fuzzy C-Means algorithm for Bias Field Estimation and Segmentation of MRI Data," *IEEE Transactions on Medical Imaging 21* , vol. 3, no. 10.1109/42.996338. PMID 11989844, pp. 193-199, 2002.

[61] N. A. Mohamed, "Modified Fuzzy C-Means for medical segmentation," M.Sc. Thesis, Univ. Louisville, KY, Louisville, 1999.

[62] A. W. Moore, "K-Means and Hierarchical Clustering Lecture Notes".

[63] H. Larochelle, Y. Bengio, J. Louradour and P. Lamblin, "Exploring Strategies for Training Deep Neural Networks," *Journal of Machine Learning Research 1*, vol. 10, pp. 1-40, 2009.

[64] B. C. M. Fung, K. Wang and M. Ester, "Hierarchical Document Clustering," 2004. [Online]. Available: http://www.cs.sfu.ca/~ester/papers/Encyclopedia.pdf.

[65] *"A Google DeepMind Algorithm Uses Deep Learning and More to Master the Game of Go | MIT Technology Review"*. MIT Technology Review.

[66] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: AN efficient data clustering method for very large databases".

[67] M. E. Otey, A. Ghoting and S. Parthasarathy, "Fast distributed outlier detection in mixed-attribute data sets," *Data Min. Knowl. Disc. 12*, Vols. 2-3, pp. 203-228, 2006.

[68] B. Zerhari, A. A. Lahcen and S. Mouli, "Big Data Clustering: Algorithms and Challenges," in *Conference: International Conference on Big Data, Cloud and Applications BDCA'15*, Morocco, 2015.

[69] X. Song, M. Wu, C. Jermaine and S. Ranka, "Conditional anomaly detection," *IEEE Trans. Knol. Data Eng. 19*, vol. 5, pp. 631-645, 2007.

[70] A. Singh, "Comparison of Machine Learning Algorithms," [Online]. Available: http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/MLAlgo_Comparisons.pdf.

[71] Y. bengio and O. Delalleau, "Justifying and Generalizing Contrastive Divergence," *Neual Computation*, vol. 21, no. 6, pp. 1601-1621, 2009.

[72] Mathworks. [Online]. Available: http://www.mathworks.com/solutions/machine-learning/examples.html;jsessionid=04d98b8343a926671cdf6fce4f29?file=/products/demos/machine-learning/swiss_roll/swiss_roll.html.

[73] J. Turian, J. Bergstra and Y. Bengio, "Quadratic Features and Deep Architectures for Chunking," *North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT)*, pp. 245-248, 2009.

[74] I. Jolli_e, Principal Component Analysis, Wiley Online Library, 2005.

[75] T. K. Marks and J. R. Movellan, "Diffusion networks, product of experts, and factor analysis. In T.W. Lee, T. -P. Jung, S. Makeig, & T. J. Sejnowski (Eds)," *Proc. Int. Conf. on Independent Component Analysis*, pp. 481-485, 2001.

[76] U. N. Raghavan, R. Albert and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *arXiv*, vol. 1, no. 0709.2938, 2007.

[77] X. Zhu, "Semi-supervised learning Literature Survey," 2008.

[78] Sklearn, "Sklearn Python," [Online]. Available: http://astroml.org/sklearn_tutorial/.

[79] Buitinck et al., API design for machine learning software: experiences from the scikit-learn project, 2013.

[80] H. Cecotti, "Active graph based semi-supervised learning using image matching: Application to handwritten digit recognition," *Pattern Recognition Letters 73*, pp. 76-82, 2016.

[81] A. Y. Ng, "Convolutional Neural Network," [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.

[82] "Support Vector Machines," 2015. [Online]. Available: https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/.