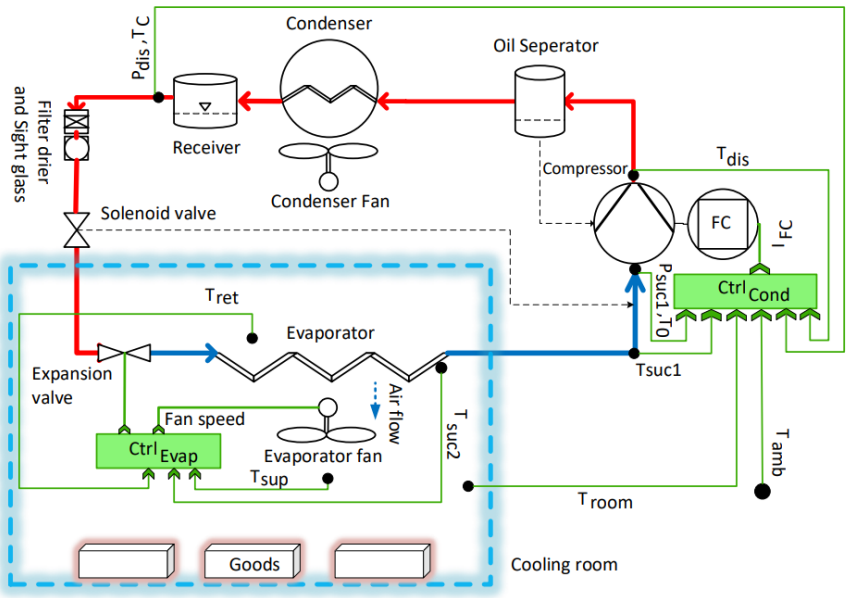
Fault Detection in Supermarket Refrigeration Systems Using Machine Learning

Master Thesis

CA10-1038



Aalborg University
Electronics & IT
Frederik bajers vej 7B
9220 Aalborg Øst, DK

AALBORG UNIVERSITY
DENMARK

Third academic year at
The Technological-Scientific Faculty
Electronics & IT
Frederik bajers vej 7B
9220 Aalborg Øst
http://www.aau.dk

Title:
Fault Detection in Supermarket Refrigeration Systems Using Machine Learning

Theme:
Control Engineering

Project:
CA4-Project

Project period:
February 2021 - June 2021

Project group:
1038

Participants:
Antonio Javier Martin Garcia
Gowsikan Sathiyaseelan

Supervisors:
Jan Dimon Bendtsen,
John-Josef Leth &
Zahra Sadat Nourbakhsh Soltani

Circulation: Digital Published
Pages: 85 without formalities
Appendix: 15
Completed: 03-06-2021

Synopsis:

Supermarket refrigeration systems are responsible for maintaining a constant temperature to preserve the quality of the goods. Failure in these appliances can lead to economical losses and food spoilage. However, manually monitoring the systems is inconvenient and expensive. The objective of this thesis is therefore to use fault detection for determining the status of the systems and predict the failure before changes occur in the system.
To solve this problem, a Support Vector Machine classifier (SVM) has been developed, using data from a simulation, to predict the state of the system in real-time. Prior to the classification, the data has been preprocessed by reducing its dimensions to facilitate the work of the classifier using Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).
Several tests have been carried out to verify the performance of the developed methods under different initial conditions and faults in the system. It can be concluded from the results that LDA is the superior method for dimensionality reduction in this application and SVM is a feasible solution for automatic fault detection having close or equal to 100% accuracy in most tests.

Preface

This thesis is written by two students at Aalborg University in their 4^{th} semester of Control and Automation Master's degree. The focus of the thesis is fault detection for supermarket refrigerant systems using machine learning methods.

The report's structure is build up of 9 chapters, starting with an introduction and ending with a conclusion, discussion and future works. All the literature and sources used throughout the report can be found in the bibliography, along with a list of figures and tables. The appendix can be found at the end of the report, containing the MATLAB codes and a table of the newly simulated data parameters used for the tests.

Special thanks to Jan Dimon Bendtsen, John-Josef Leth and Zahra Sadat Nourbakhsh Soltani for their supervision and guidance throughout this period. Additionally, thanks to Bitzer Electronics A/S for providing the data and simulation necessary for this thesis.

Antonio Javier Martin Garcia
amarti19@student.aau.dk

Gowsikan Sathiyaseelan
Gsathi16@student.aau.dk

Mathematical notation

In order to encourage a proper understanding of the mathematical contents, consistency of the notations has been kept throughout the report. Furthermore to avoid confusion with normal letters all mathematical symbols are written in *italic*. A list of all the symbols used throughout the whole report can be found in Table 0.1 below.

Table 0.1: List of all the mathematical symbols and their description

Symbol	Description
x	Lowercase variables are vectors
X	Uppercase variables are matrices
$\bar{x}$	A variable with a bar is the mean of said variable
D	Number of dimensions
M	Number of dimensions the data is reduced to
N	Number of samples
N_{Sv}	Number of support vectore
$n = 1, 2, ..., N$	Subscript n is the sample index
$i = 1, 2, ..., D$	Subscript i is the dimension index
x^T	Superscript T is the transpose of x
σ	Standard deviation
z	Standardised data
S	Covariance matrix
u	Eigenvector
U	Transformation matrix
λ	Eigenvalues
μ_1,μ_2	Mean of the different classes
S_B	Between class covariance matrix
S_W	Within class covariance matrix
w	Eigenvector in LDA method
$J(w)$	Function for maximising the separation and minimising the variance
Y	Projected data
W	Transformation matrix
w	Coefficient of the line / weight vector of the SVM hyperplane
S_v	Support vectors
b	Bias of SVM hyperplane
L	Lagrangian
t	Target value
a	Lagrangian multiplier
ξ	Slack variable
$x*$	New observation
C	Uppercase C determines the restriction of SVM
c	Lowercase c is the number of classes
K	Kernel function

Acronyms

AAU Aalborg University

CNN Convolutional Neural Network

DAGSVM Directed Acyclic Graph SVM

FDA Flexible Discriminant Analysis

LD Linear Discriminant

LDA Linear Discriminant Analysis

MANOVA Multivariate Analysis of Variance

PC Principal Components

PCA Principal Component Analysis

P-H Pressure Enthalpy

PoV Proportion of Variance

QDA Quadratic Discriminant Analysis

RDA Regularized Discriminant Analysis

SRS Supermarket Refrigerant System

SSS Small Sample Size

SVD Singular Value Decomposition

SVM Support Vector Machine

Contents

CHAPTER 1

Introduction

Refrigerated goods are greatly influenced by the temperature they are stored at, which is why inconsistencies in the temperature can reduce the quality of the products and lead to food spoilage. A Supermarket Refrigeration System (SRS) is therefore an important appliance used to keep the goods from going bad. However, it can occasionally fail at times, which can lead to economical losses for the stores and have health issues for the customers who consume these products. The problems commonly occurring on SRSs can be from faults in the condensing unit or evaporator side and in some cases be caused by evaporator fan faults which are due to either reduced speed or a defective fan. These issues can be detected by implementing automatic fault detection on the SRS which would give an early warning before the goods are affected. Early fault detection would allow for the products to be transferred to another cold storage unit or for a technician to arrive and solve the issue before it affects the temperature. This way the loss can be kept at a minimum and the problem can be solved quickly.

A SRS without any automatic fault detection has to be manually monitored by measuring its temperature to detect whether the system is faulty. This type of diagnosis and detection is undesirable as it is heavily dependent on humans to check whether the temperature deviates from the set point. It is an unreliable approach as it requires to be periodically monitored by someone to ensure that a faulty system does not go unseen. Even if a faulty system is detected using this method, it would still be too late as the integrity of the products have been compromised due to the temperature difference in the storage units.

This project was proposed by Bitzer Electronics, a refrigeration and air conditioning company that has been working in the field for 86 years. Their goal is to ensure reliable and efficient systems for their customers. Bitzer Electronics also specialises in other areas of the cold chain such as reefer containers and reefer trucks which are

used throughout the whole delivery process of goods. They also produce household refrigeration systems and controllers for different industrial refrigeration systems. Ecostar is the condensing unit they produce for SRSs. The company's other objective is in the development of a proper automatic fault detecting model. As manual human detection is costly and time-consuming.

The main focus of this thesis is therefore in the development and implementation of an automatic fault detection method for SRSs as it is relevant for having a reliable system. The goal is to have a method which is applicable for a variety of SRSs as they can differ in sizes, components and capacities depending on the supermarkets' requirements. The weather conditions can also have an affect on the condensing units. It is therefore desired to have a method which is robust enough to take all of these factors into account.

The availability of data from SRSs encourages the usage of feature extraction and dimensionality reduction methods for analysing which parameters are most informative for fault detection. For this purpose, machine learning methods such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) can be implemented. In addition, the company has provided an ECOSTAR simulation model which can be altered by changing its parameters to represent different fault scenarios in the system. The extracted data from the model can then be used to identify which feature extraction method is best suited for finding faults for a wide range of systems.

It is also necessary to use a method for classifying the data. As the feature extraction methods only show the most prominent variables. One of the methods which could be used for the classification is a Convolutional Neural Network (CNN) which is based on deep learning. Meaning that it can be trained to recognise the difference between a faulty and non-faulty system. The other method involves Support Vector Machine (SVM) which requires less data than a CNN for classification as it separates the data from different classes and can thereby distinguish between a faulty and non-faulty system. In this thesis it has been chosen to work with SVM for classifying the data.

1.1 Problem Statement

The intended outcome of this thesis is to develop and implement automatic fault detection for a SRS.

The current issue with a SRS is that it is too expensive to manually monitor all the variables from the system. It is therefore only reasonable to monitor the variables which directly affect the temperature. This causes a late detection of malfunction in the system which compromises the quality of the goods. It is therefore instead desired to detect any faults as early as possible using the collected data from the different sensors throughout the system. As it is imperative to maintain a consistent temperature throughout the whole system in order to avoid food spoilage and keep the goods in their optimal condition.

The end goal of this project is to use and compare different feature extraction methods on data from the system to analyse which of them is best suited for extracting important features for classifying a faulty SRS. The most commonly used methods for this purpose are PCA and LDA which are known for their dimensionality reduction. Reducing the data is a necessity as the fault detection needs to work in real-time, since processing a lot of data would be time-consuming due to heavy computations. The reduced data can then be classified using SVM.

CHAPTER 2

System Overview

The following chapter describes the refrigeration process in detail and explains its relation to the P-H diagram. Furthermore, the data and simulation received from Bitzer Electronics are also described.

2.1 Description of the refrigeration process

In order to maintain adequate conditions for food in supermarkets, it is necessary to have precise control of the temperature and environment in which they are stored. Inadequate temperatures can lead to food degradation, energy expenditure, etc.

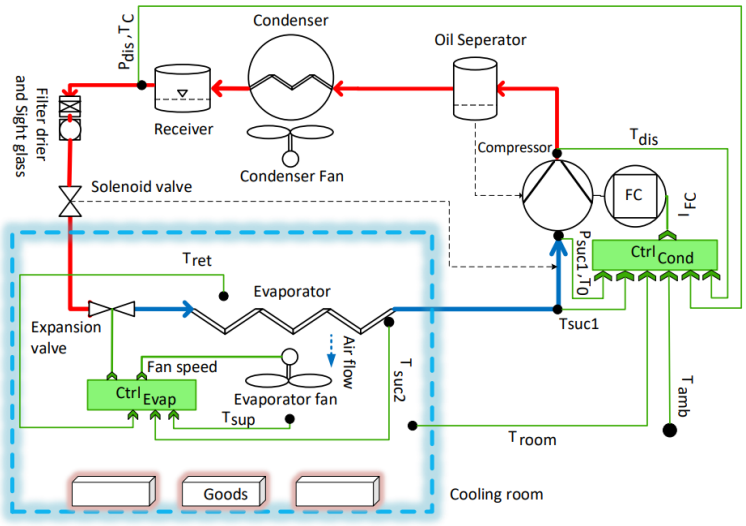


Figure 2.1: Schematic of a refrigeration system used [1]

Refrigeration systems based on heat transfer are used to control the temperature in supermarkets. They can vary from one system to another, however, they normally work by extracting the heat from the cold room and expelling it to the environment [1] [2]. For this process, the refrigerant liquid changes its state between liquid and vapour in the different phases that take place: evaporation in the cold room and condensation during dissipation to the outside. An overview of the SRS by Bitzer Electronics can be seen in Figure 2.1. It is composed of two main parts; a condensing unit and an industrial evaporator in the cold room, each with its corresponding controller: $Ctrl_{Cond}$ and $Ctrl_{Evap}$ respectively. The first controller is connected to a series of sensors allowing for control over the speed of the condenser fan and compressor (V_{cpr}), which controls the temperature in the cooling room.

The second controller is in charge of regulating the evaporator fan and expansion valve. Depending on the opening degree of the expansion valve the flow of cooling liquid can be increased or decreased allowing for more or less transfer of heat from the room to the liquid. Increasing the evaporator fan speed allows for more airflow to the evaporating surface and thereby produces more heat transfer.

Failures in this section of the system, more specifically in the evaporator fan, can lead to uneven temperatures in certain areas of the cold room, which can compromise the quality of the goods. When the evaporator fan malfunctions, it implies that there is a reduction in the airflow and consequently a reduction in heat transfer. Faced with this situation, the $Ctrl_{Evap}$ increases the temperature difference between the coolant and the air, so that the desired room temperature is maintained. This has a series of effects on some of the measured variables, such as suction pressure (P_{suc}), vapour density (ρ_v), etc. which results in the compressor having to work harder to maintain the volumetric flow constant. This can be analysed from certain variables which can determine when a fault is occurring, but not all the variables are always accessible, so a different selection of the key features are necessary depending on the refrigeration system parameters available.

2.2 P-H diagram

The pressure enthalpy diagram (log P-H diagram) is a graphical representation of states of fluids showing an overview of their behaviour under different conditions. It is a valuable diagram specifically made for different refrigerants from which thermal-dynamical properties can be deduced.

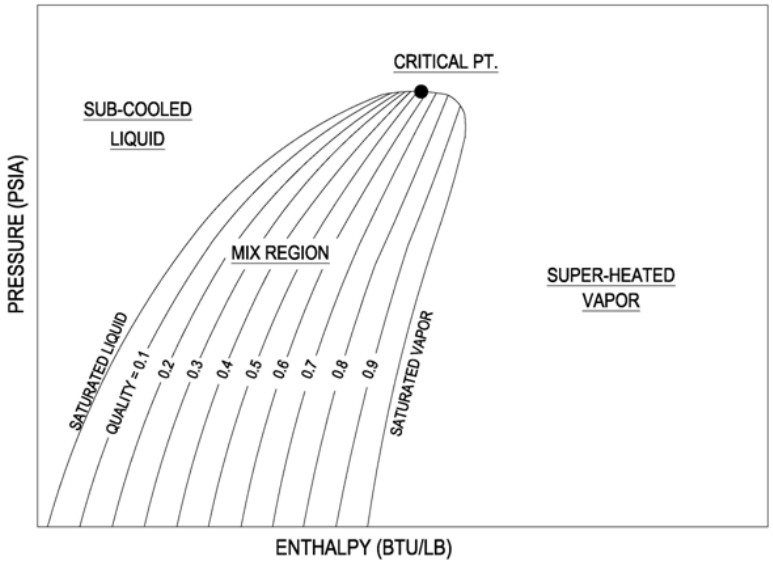


Figure 2.2: Illustation of the different phases on a P-H diagram [3]

A P-H diagram is a representation with a y-axis of absolute pressure and a x-axis of specific enthalpy. The enthalpy is typically measured in joule per kilograms (J/Kg) or Btu/lb while the pressure is in Pascal (Pa) units or pounds per square inch (psi). The P-H diagram contains a mixed region stretching from a saturated liquid on the left to a saturated vapour on the right as seen in Figure 2.2. The point they meet at the top is known as the "critical point" which is where the refrigerant can experience the highest temperature and remain in a liquid state. If the temperature increases beyond the critical point the refrigerant would only exist in a vapour state regardless of the pressure [4].

The diagram can be seen as a map indicating a refrigerants different thermodynamic states. The left side of the saturated liquid line has a low specific enthalpy indicating an area where the fluid is in a liquid phase (sub-cooled liquid). However, to the right of the saturated vapour line the enthalpy is high and the fluid is in a vaporous state (super-heated vapour). The mixed region delimited by the saturated liquid and vapour lines describe a mixture of both states. Meaning that most of the left side of the mixed region is in a liquid state while only a small part is in a vaporous

state. While vice versa for the right side of the mixed region as it is predominantly in a vaporous state with a small amount being in a liquid state. The horizontal lines inside the mixed region seen in Figure 2.2 indicate the constant vapour content known as “quality” which is a measure of the ratio of vapour mass and the total mass. In other words, if a refrigerant moves from left to right inside the mixed region the quality increases and the fluid is evaporating. Whereas if it is moving from right to left the refrigerant is condensing and the quality decreases [3].

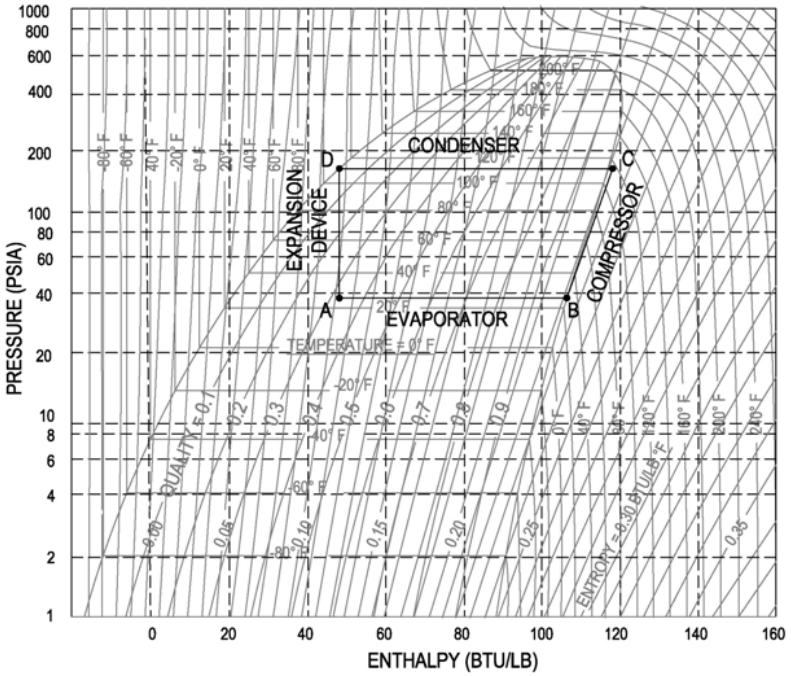


Figure 2.3: The refrigeration cycle on a P-H diagram [3]

The lines of constant temperature (isothermal lines) are vertical in the sub-cooled liquid region, horizontal in the mixed region, and drop steeply towards the enthalpy axis in the superheated vapour region. The constant pressure lines (isobaric lines) are parallel to the x-axis which can be seen in Figure 2.3.

A refrigeration cycle generally consists of four major components evaporator, compressor, condenser and expansion valve. At point “A” in Figure 2.3, the refrigerant is in a partial liquid-vapour state where the temperature, pressure and enthalpy are all low. To move from point “A” to “B” the evaporator transfers heat to the refrigerant converting the remaining liquid to vapour. The ideal evaporator coverts 100 % of the liquid to vapour and nothing more as the refrigerant would be superheated if the temperature increases beyond that which is undesirable as the temperature and pressure should remain constant while only the enthalpy should increase in the evaporator.

The refrigerant vapour at point “B” has a lot of heat but with a low temperature and pressure, so in order to go from point “B” to “C” the refrigerant is compressed using a compressor which increases the temperature and pressure without adding any heat which is known as an “adiabatic process”.

At point “C” it is a hot, high-pressure refrigerant vapour. Once the refrigerant goes from point “C” to “D”. It is first cooled through the condenser which coverts the vapour to a liquid by dispensing the heat. If the temperature decreases the refrigerant ends up being sub-cooled which is not ideal for the condenser. It is desirable to remove the same amount of heat added by the evaporator process while still retaining the same temperature and pressure.

The last step in the refrigeration cycle is the expansion valve which main goal is to lower the pressure, and in turn make it possible to go from point “D” to “A” again as the temperature decreases along with the pressure drop while the enthalpy stays the same. Thereby ending up with a cold refrigerant. This is the theoretical explanation of the refrigeration cycle which might differ in real life due to limitations of equipment’s [5].

2.3 Data from Bitzer Electronics

In this section, there will be a description of the data provided by Bitzer Electronics. This data has been collected using the simulation described in Section 2.4, modifying certain parameters from one experiment to another. All the variables used for the simulation will be explained in this section including the ones used to develop the proposed methodology in Section 1.1. The main features from this data will be extracted through PCA / LDA dimensionality reduction methods and subsequently be used for training the SVM classifier.

The folder provided by Bitzer Electronics contains a total of 115 different experiments, with their corresponding figures in which a total of 10 different variables are shown. An example of one of these experiments, showing the variables can be seen in Figure 2.4.

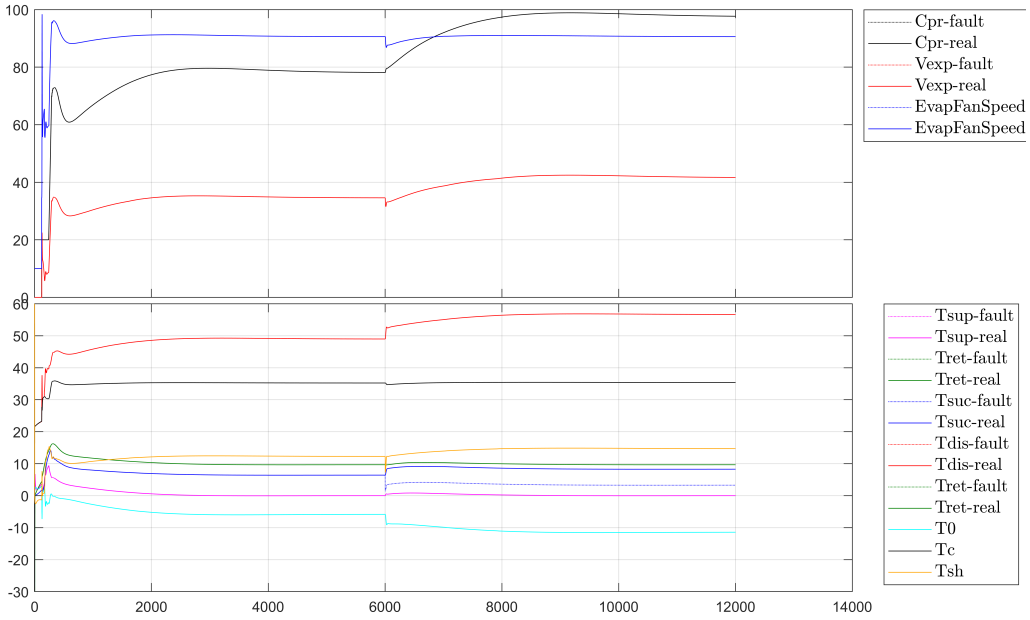


Figure 2.4: Example of a figure with the data provided by Bitzer Electronics, showing Non-Faulty and Faulty operating data

In this figure a total of 19 variables are represented, however, only 10 different variables can be distinguished. This is because in the representation of the variables, those that share the same name but are distinguished in the labels (for example, $Cpr-fault$ / $Cpr-real$), are superimposed on the graph. In this project, only the variables labelled as "$fault$" have been used, including the variables without any second label (T_0, T_c, etc.), since they are not superimposed.

Each variable contains data of both a working and faulty system. As the first 6000 observations (from sample 1 to 6000) belong to a correctly functioning system, while the next 6000 samples (from sample 6001 to 12000) belong to a faulty system. This can be seen in Figure 2.4, where after sample 6000 a change in the trend is observed for all the variables.

The files generated by the simulation contain a struct variable with all the variables which interact with the simulation. This struct consists of data from both the simulated variables, as well as useful information of the different tests performed. This information includes the test number, ambient temperature, $HeatLoad$, temperature setpoint (T_{set}), start and end samples for the tests, as well as the start sample for the faulty data, labels for the faulty/non-faulty samples, error message and a time variable.

Apart from this, the remaining variables include the $Offset/Scale$ for the different faults and the data for the variables. Due to some of these variables being superimposed, as mentioned before, only the following variables will be studied from the simulations: T_{ret}, T_{suc}, T_{dis}, T_0, T_{sh}, T_c, T_{sup}, P_{suc}, P_{dis}, Cpr, V_{exp}, $EvapFanSpeed$, $CondFanSpeed$.

The 115 different tests are divided into groups of 6 tests for each faulty signal, varying in the $HeatLoad$ (1000, 2000, 4000, 13000, 17000, 20000), T_{set} (0, 7, 12) and $Offset/Scale$ values as seen in Table B.1. This table is however for the newly simulated data but the same principle applies with the only difference being the $Offset/Scale$ values and the sample size.

The $HeatLoad$ parameter represents the amount of work done by the compressor. Figure 2.5 shows a comparison between tests with a low and high $HeatLoad$ parameters. A value of $O(10^3)$ leads to results with very large oscillations in the variables compared to data obtained with values of $O(10^4)$.

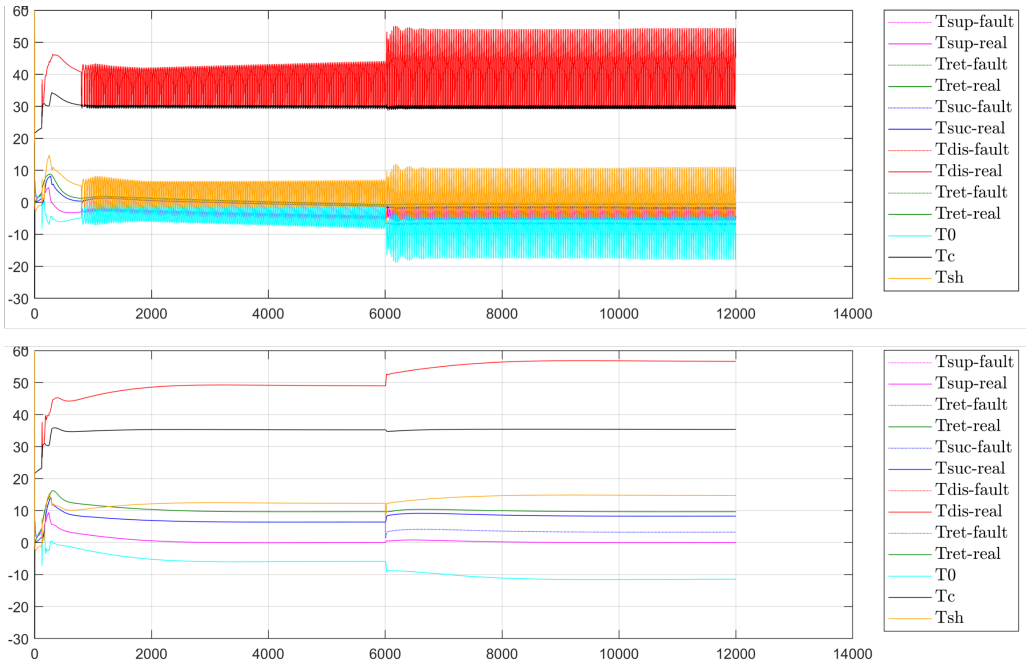


Figure 2.5: Comparison of two datasets with different heatload parameters. The top subplot being at 1000 and the bottom subplot being at 13000

2.4 Simulation

Bitzer Electronics has provided a simulation model of their "ECOSTAR" system, which is one of their supermarket condensing units. The model is specifically made for one of their available "ECOSTAR" models but can however be altered to produce data representative of a variety of different models. The simulation has been prepared to simulate a wide range of faults making it possible to produce both faulty and non-faulty data for different areas of the system.

The simulation model is made in MATLAB using Simulink as seen in Figure 2.6. Allowing the user to change certain parameters and plot a simulated diagram illustrating how the data from the sensors would look like under the specified conditions similar to Figure 2.4 in the previous section.

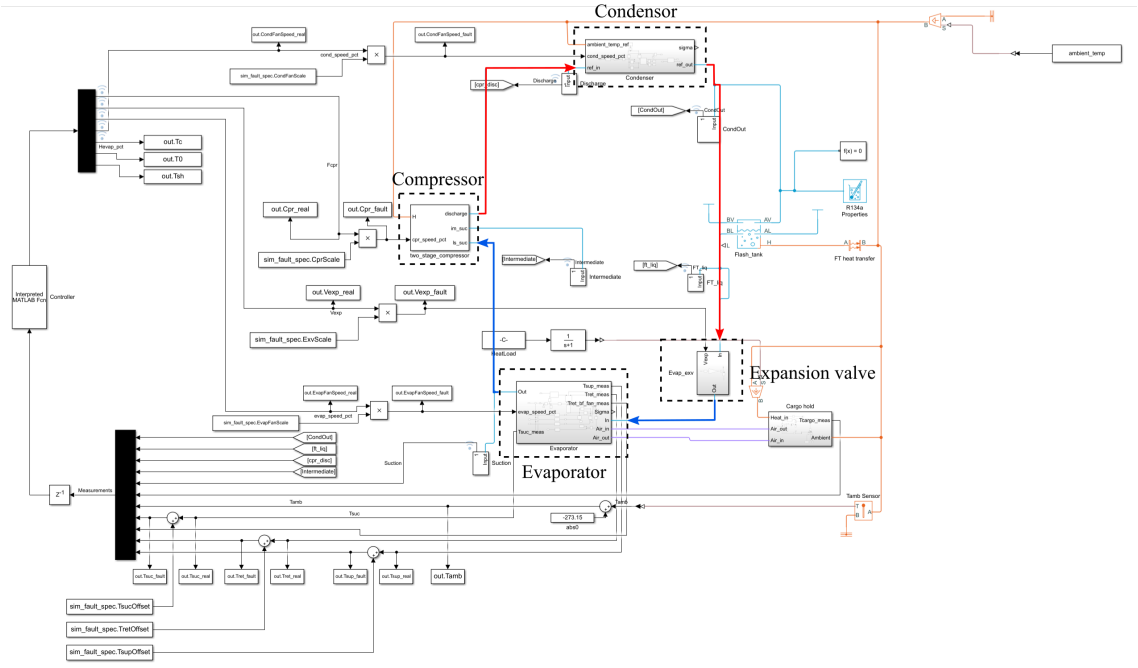


Figure 2.6: The simulation model used in MATLAB

The way the simulation works is by using a MATLAB-script which takes an excel file as input for the system. This file contains all the parameters which are configurable by the user. The first couple of variables are related to the simulation plot ($Start_{Sample}$, $Fault_{Sample}$, End_{Sample}) and the general system itself (T_{set}, T_{amb}, $HeatLoad$). While the last variables are related to the faults which can be simulated using the model ($T_{sucOffset}$, $T_{retOffset}$, $T_{disOffset}$, $P_{disOffset}$, Cpr_{Scale}, Exv_{Scale}, $EvapFan_{Scale}$, $CondFan_{Scale}$). These faults are either temperature/pressure offsets or malfunctioning/underperforming parts in the system, which can be due to fan

faults, blocked valve, lack of refrigerant or wear over time. Once the system parameters have been specified in the file the simulation can be run. The output data from that test is then collected in a “struct” and plotted in a graph as mentioned in Section 2.3.

The simulation model gives the user the freedom to specify the faults and the severity of these faults which can be beneficial when creating a system for fault detecting especially when the real SRS is not accessible. An explanation of all the variables used in the simulation and the data from Bitzer Electronics can be found in Table 2.1. Instead of using the data provided by Bitzer Electronics new simulations have been created based on more realistic offsets and scales which can be seen in Appendix Table B.1.

Table 2.1: List of symbols with their description and units.

Symbol	Description	Unit
T_{set}	Setpoint temperature	$[°C]$
T_c	Condensing temperature	$[°C]$
T_{sh}	Superheat temperature	$[°C]$
T_{room}	Cooling room temperature (sensor)	$[°C]$
T_{amb}	Ambient temperature (sensor)	$[°C]$
T_{suc}	Suction temperature (sensor)	$[°C]$
T_0	Evaporation temperature (sensor)	$[°C]$
T_{dis}	Discharge temperature (sensor)	$[°C]$
T_{ret}	Returned air temperature (sensor)	$[°C]$
T_{sup}	Supplied air temperature (sensor)	$[°C]$
P_{suc}	Suction pressure (sensor)	$[bar]$
P_{dis}	Discharge pressure	$[bar]$
I_{FC}	Converter current	$[A]$
FC	Frequency converter	$[-]$
$Ctrl_{Evap}$	Evaporator controller	$[-]$
$Ctrl_{Cond}$	Condenser controller	$[-]$
Cpr	Compressor	$[-]$
V_{exp}	Expansion valve	$[-]$
$EvapFanSpeed$	Evaporator fan speed	$[-]$
$CondFanSpeed$	Condenser fan speed	$[-]$
$HeatLoad$	Compressor work	$[W]$

CHAPTER 3

Theory

This chapter describes the theory behind the different dimensionality reduction and classification methods. Each of the dimensionality reduction methods are divided into steps, key issues and adaptations. Continuing with a comparison between both methods. Lastly the chapter ends with a description of the classifier for both linear and non-linear cases.

3.1 PCA

Principal Component Analysis (PCA) is a data analysis method used to emphasise variation and bring out patterns in data. It transforms data into a lower dimension while maintaining the main characteristics of it. It reduces the data by geometrically projecting them onto lower dimensions known as Principal Components (PC's) and tries to find the best description of the data using as few PC's as possible.

The first PC, which accounts for the most variation, is chosen by minimising the total distance of the data and their projection to the PC. Minimising the distance to the PC also means that the variance of the projected points is maximised (σ^2) [6]. The second and subsequent PC's are selected with a similar criteria, however they are chosen to be uncorrelated to the previous PC's and therefore orthogonal. This requirement means that the maximum number of PC's possible is the minimum between number of samples or number of features.

This method also maximises the correlation between data and their projection, which is similar to carrying out multiple linear regressions on the projected data against each variable of the original data.

The PC's are defined as linear combinations of the original data variables. The PCA can be interpreted as a rotation matrix that rotates the data in such a way that the projection with the greatest variance goes along the first axis. The new variables depend on the dataset, rather than being pre-defined functions. This means that the PC's are adaptive to each individual case. PCA is an unsupervised learning method which is similar to clustering. Clustering is a method of unsupervised segmentation and grouping, since the groups obtained are not known prior.

3.1.1 Steps

The Principal Component Analysis can be carried out in 5 steps [7]:

Step 1: For all the data to influence equally, the first step of conducting PCA is to standardise all the values. This prior step is critical as the method is sensitive towards the variance of the initial variables. Meaning that the variables with a greater variance will have a greater influence and effect on the method. A variable with a range between 0 and 100 will have a greater influence than one between 0 and 10, leading to inaccurate results. Having a set of variables $x = x_1, x_2, \ldots, x_N$ (with N being the number of samples) which belongs to a D-dimensional space. The standardisation is normally done with a simple formula: the new value (z_n) is obtained by subtracting the mean ($\bar{x}$) and dividing by the standard deviation (σ) for each variable (x_n) with $n = 1, 2, \ldots, N$. With this step, all the variables will be on the same scale.

$$z_n = \frac{x_n - \bar{x}}{\sigma} \tag{3.1}$$

Where, $\bar{x}$ is the mean of each variable.

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{3.2}$$

This step is not necessarily required if the original dataset has the same range in all dimensions, as it will not have a big effect on the PCA.

Step 2: Covariance matrix computation. This step serves to see the relationship that exists between the variables, comparing how much each initial variable differs from the mean. In this way, the variables that have a high correlation can be

detected, as they only provide little additional information to the already existing one. To detect this correlation, the covariance matrix S is computed.

$$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T \tag{3.3}$$

S is a symmetric $D \times D$ matrix (where D is the number of dimensions). In the diagonal it can be found the variances of each variable (since the covariance of a variable with itself is its variance) and since the covariance is commutative there is a symmetry with respect to the main diagonal.

Step 3: Compute the eigenvectors and eigenvalues of the covariance matrix to identify the Principal Components.

Principal Components are new variables that are constructed as linear combinations of the initial variables. These combinations are done in such a way that the PC's are uncorrelated and most of the information within the initial variables are compressed into the first components. This allows selecting the components that contain the maximum information, discarding those that have little to add. In geometric terms, Principal Components represent the directions of the data that have a maximal amount of variance. The larger the variance carried by a line, the larger the dispersion of the data points along it, thus the more information it has.

This can be seen in Figure 3.1. The PC's are chosen according to one of the two criteria that are equivalent: comparing the variance and looking for the greatest possible dispersion of data (blue dots) projected along the PC (red dots), or comparing the distance of the data to the PC (red lines) and looking for the minimum possible distance, in absolute value, for all the samples.

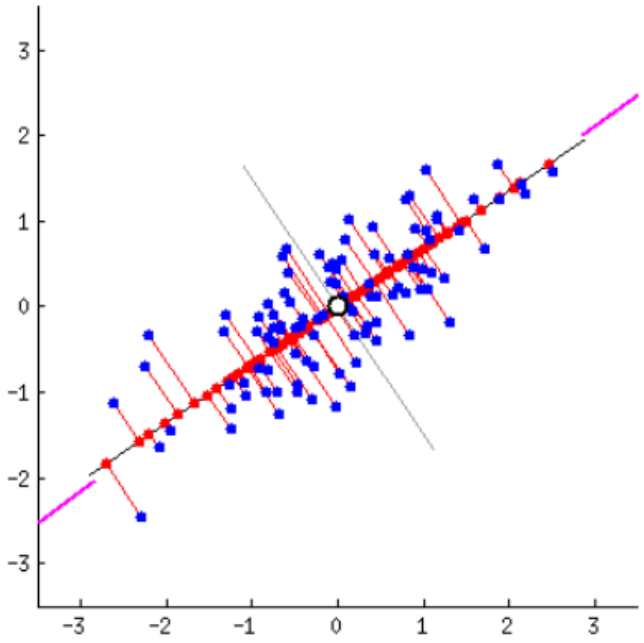


Figure 3.1: First Principal Component in a 2 Dimensional data [7]

If the dataset is defined as D-dimensional vectors $x_1, \ldots, x_N$ or, equivalently a $D \times N$ data matrix X, where the x_j is the row vector of observations of the jth variable (Dimension). The goal is to find a linear combination of the rows of X that maximises the variance (dispersion along it). This linear combination can be represented as $\sum_{j=1}^{D} u_j x_j = u^T X$, where u is a column vector of constants $u_1, u_2, \ldots, u_D$.

The variance is then, $var(u^T X) = u^T S u$, where S is the covariance matrix. Finding the linear combination with maximum variance is now equivalent to obtaining the vector u that maximises the quadratic form $u^T S u$ [8]. Furthermore, an additional constraint has to be imposed, the vector u has to have norm one, this is, $u^T u = 1$. Now the problem can be reformulated as trying to maximise $u^T S u - \lambda(u^T u - 1)$, where λ is a Lagrange multiplier.

$$Su - \lambda u = 0 \Leftrightarrow Su = \lambda u \tag{3.4}$$

Vector u is therefore the eigenvector, and λ is the corresponding eigenvalue of the covariance matrix S.

The eigenvectors of the covariance matrix are the directions of the axes where there is the most variance. The eigenvalues give the amount of variance carried in each

Principal Component. The PC's are obtained by sorting the eigenvectors from highest to lowest using the eigenvalues associated with them. This can be represented in a scree plot as seen in Figure 3.2, where 10 eigenvalues have been sorted and normalised.

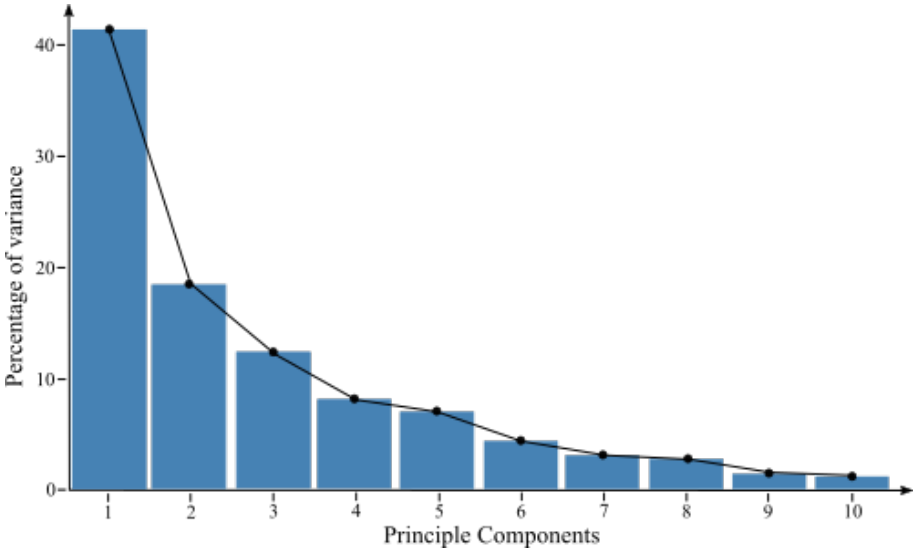


Figure 3.2: Scree plot of the eigenvalues of a 10 dimensions dataset [7]

Step 4: Feature vector. Once they have been sorted, the feature vector needs to be formed which is a matrix composed of the eigenvectors carrying most of the information about the original dataset. The dimension of this matrix determines the final dimension of the transformed data.

The problem is: how to choose the necessary number of eigenvectors without a great loss of information? If D the total number of original dimensions of eigenvectors/eigenvalues and M ($M < D$) the number of dimensions to which it is going to be reduced. To estimate the amount of information that the eigenvalues encompass, the Proportion of Variance (PoV) formula can be used:

$$PoV(M) = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_M}{\lambda_1 + \lambda_2 + \cdots + \lambda_M + \cdots + \lambda_D} \tag{3.5}$$

when λ_i are sorted in descending order it is safe to stop when $PoV > 0.9$.

Step 5: Recast the data along the axes of the Principal Components. Using the feature vector formed in the previous step, the original data is transformed and reoriented from the original axes to the ones that correspond to the PC's. In order

to project the data into M number of PC's Equation 3.6 has to be computed, where U is the $D \times M$ transformation matrix or feature vector containing the M eigenvectors.

$$Y = U^T X \tag{3.6}$$

3.1.2 Key issues with PCA

Covariance and correlation matrix: As mentioned in Section 3.1.1 step 1, it is very important that the scale of all variables are the same. This is due to the influence that different scaling has when it comes to studying their variance and obtaining an adequate covariance matrix. Since the covariance matrix of a standardised dataset is the correlation matrix R of the original dataset, a PCA on the standardised data is also known as a correlation matrix PCA [8]. The eigenvectors of the correlation matrix now define the uncorrelated maximum variance of the standardised variables. These new vectors do not correspond to those calculated prior to standardisation. The variance encompassed by each of the PC's chosen in this way do not coincide either. Normally a greater number of PC's with the standardisation is required than without it to encompass the same percentage of variance. The Singular Value Decomposition (SVD) approach is also valid in this context [8]. An SVD procedure of the standardised data matrix is equivalent to a PCA correlation matrix of the dataset. Unlike the covariance matrix PC's, correlation matrix PC's do not vary based on linear changes in units of measurement and, therefore, are the appropriate choice for datasets where there are different scales for each variable.

Centring the data: On some occasions, PCA is similar to an SVD of a row-centred data matrix. In certain applications, centring rows of the data matrix may be inappropriate. In these situations it may be better to avoid any preprocessing of data and subject the matrix of unfocused data to an SVD or auto decompose the matrix of second non-centred moments, whose eigenvectors define linear combinations of the uncentered variables.

3.1.3 Adaptations of Principal Component Analysis

There are several adaptations to the original PCA method [8]. These are among the most common that can be found: Functional Principal Component Analysis, Simplified Principal Components, Robust Principal Component Analysis, Symbolic Data Principal Component Analysis, etc.

3.2 LDA

Linear Discriminant Analysis (LDA) is another dimensionality reduction method commonly used for supervised data classification. It is similar to PCA but instead focuses on maximising the separability between classes. LDA reduces higher dimensional data by projecting it onto a lower dimensional space by maximising the separation of the data points while simultaneously minimising the dispersion of the data, thereby achieving maximum class discrimination in the dimensional-reduced space [9]. This process results in class separation which avoids overfitting ("Curse of dimensionality") and reduces computational costs.

To explain how LDA works an example of a two-dimensional dataset can be used. As here instead of projecting the data onto one of the axes and neglecting the useful information from the other axis, LDA creates a new axis using information from both original axes and projects the data onto this new axis in a way that maximises the separation between the two classes. These new axes follow two criteria: having maximised distance between the means and minimised variance within each class which can be seen in Figure 3.3.

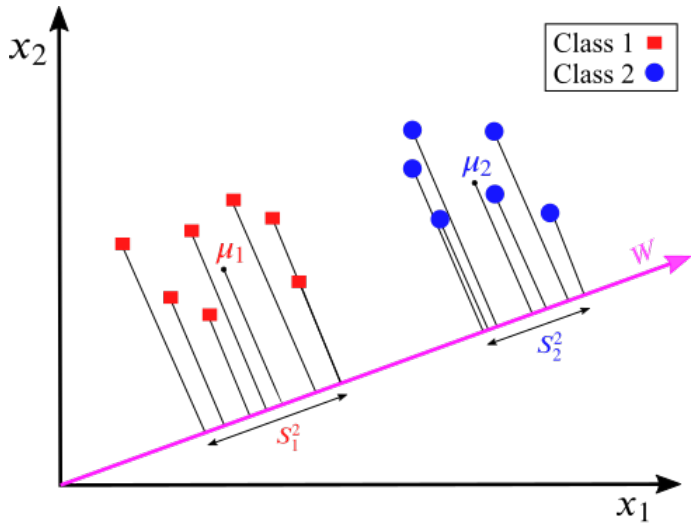


Figure 3.3: LDA maximising the class separation [10]

LDA was originally described as a two-class problem but was later generalised for multiclass classification. There exist two different types of LDA methods known as class-dependent and class-independent. In the class-dependent LDA, a separate lower dimensional space is calculated for each class to project the data onto. While in the class-independent LDA, each class is projected onto the same lower dimensional space [11]. All the data therefore share the same transformation regardless

of their class unlike the class-dependent LDA, which transforms the class data independently. It is therefore easier to use class-independent LDA for classification as a new unknown sample can be transformed using a single transformation. Whereas for the case of class-dependent LDA, it can be difficult to figure out which class transformation to use on the new sample data as the class is unknown.

3.2.1 Steps

The goal with dimensionality reduction using LDA can be achieved in four steps. In the case of two class D-dimensional reduction problems with a desired projection onto one dimension, the steps will be as follows.

Step 1: The first step is to compute the between-class covariance matrix (S_B) which is the distance between the mean of the different classes. It shows how well the data is scattered across the classes.

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{3.7}$$

Where, the mean of each class is represented as μ_1 and μ_2.

$$\mu_1 = \frac{1}{N_1}\sum_{n=1}^{N_1} x_n, \qquad \mu_2 = \frac{1}{N_2}\sum_{n=1}^{N_2} x_n \tag{3.8}$$

Step 2: The second step is to calculate the within-class covariance matrix (S_W) which is the distance between the mean and the samples in each of the classes. It shows how well the data is scattered within each class which can be calculated by finding the sum of the covariance matrix for each of the classes.

$$S_W = \sum_{n=1}^{N_1}(x_n - \mu_1)(x_n - \mu_1)^T + \sum_{n=1}^{N_2}(x_n - \mu_2)(x_n - \mu_2)^T \tag{3.9}$$

Step 3: After calculating the between-class covariance matrix and the within-class covariance matrix, the third step is to find the vector w which is the transformation vector projecting the D-dimensional data into one dimension. This is also the vector

that maximises the separation between the two classes and minimises its variance using the S_B and S_W from Equations 3.7 and 3.9 respectively in:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{3.10}$$

In order to maximise the separation, $J(w)$ must be differentiated with respect to w and made equal to zero, finding the following equality:

$$S_W w = \lambda S_B w \tag{3.11}$$

where, λ is the grouping of all scalar values. In this problem, λ has little importance since it is the direction of w that is desired to find, and not its magnitude. This can however be rewritten to Equation 3.12 if S_W has a full rank (non-singular), as it will be invertible.

$$w = S_W^{-1} S_B w \tag{3.12}$$

Since $S_B w$ follows the same direction as $\mu_2 - \mu_1$, Equation 3.12 can be rewritten as follows.

$$w \propto S_W^{-1}(\mu_2 - \mu_1) \tag{3.13}$$

The eigenvector is a non-zero vector that represents the direction of a new lower dimensional space. While the eigenvalue is a scalar value representing the scaling factor of the eigenvector. Meaning that the eigenvector represents an axis in a new lower dimensional space, whereas the eigenvalue shows the robustness of this vector. This indicates how well the eigenvector maximises the between-class variance and minimises the within-class variance.

Step 4: The fourth and last step is to reduce the original data matrix by projecting it onto the lower dimensional space.

$$Y = w^T X \tag{3.14}$$

where, Y is the projected observation and X is the original data matrix. While w is the constructed lower dimensional space using the eigenvector.

The w vector can be considered as the eigenvector which maximises the $J(w)$ function, this being, the eigenvector with the highest eigenvalue. In the case where the data is projected down onto $k > 1$ dimensional space, several eigenvectors are required in order to form the W matrix, where $W = w_1, \ldots, w_k$. This is a transformation matrix where the columns are the eigenvectors of $J(w)$. Since $J(w)$ has D eigenvectors, the eigenvectors with the largest eigenvalues are used to represent the lower k-dimensional space while the others are neglected.

And then the projection of the data will be:

$$Y = W^T X \tag{3.15}$$

3.2.2 Key issues with LDA

Even though LDA is widely known and a commonly used data reduction method it suffers from two main problems: linearity and small sample size.

Linearity problem: LDA is used to find a linear transformation that separates different classes. However, it can not find a lower dimensional space if the classes are non-linearly separable. Meaning that in the cases where the means of the classes are approximately equal to each other, the discriminatory information does not exist and the LDA will therefore not be able to find a lower dimensional space. To solve this issue a kernel method can be used which transforms the data from non-linearly separable to linearly separable as seen in Figure 3.4. This is done by transforming all the samples using a nonlinear mapping to a new feature space. Some of the different types of kernel functions used for this purpose include Polynomial, Gaussian, Radial Basis Function etc.

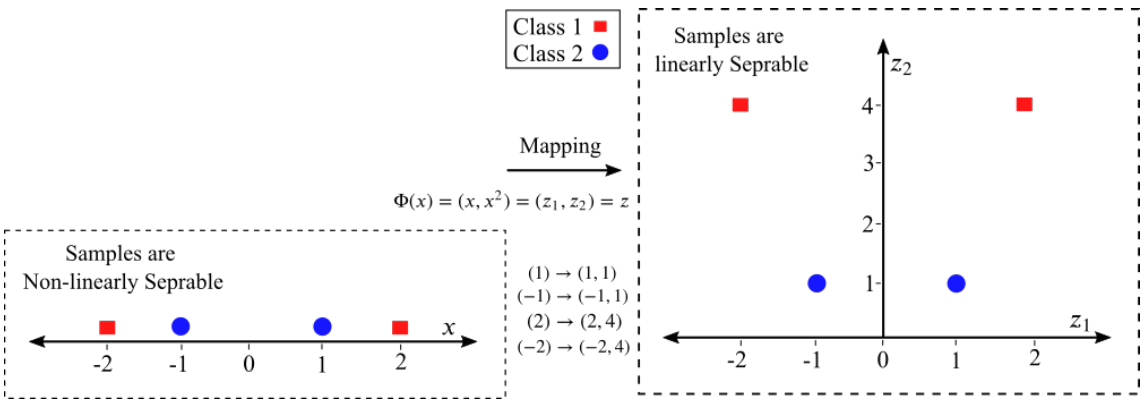


Figure 3.4: The linearity problem solved using the kernel method [12]

Small Sample Size: One of the big problems of dimensionality reduction methods is the Small Sample Size (SSS). It occurs when there is a lower number of training samples available for each class compared to the dimensionality of the sample space. To be more specific for this method, it occurs when the within-class variance is singular. Since linear discriminant analysis (LDA) requires the within-class covariance matrix to be nonsingular, LDA cannot be directly applied to SSS problems. These are some of the solution used for solving the SSS problem. The first one being Regularisation (RDA) where the identity matrix is scaled by a regularisation parameter and added onto the within-class variance making it non-singular. The problem with this method is choosing the right regularisation parameter as it could lead to poor performance. Another method to solve the issue is using sub-space in which the original data is reduced using a non-singular lower space. The drawback of this method is that some discriminant information is lost. A third solution for solving the issue is using Null-space. Here the idea is to remove the null space of the within-class variance. However in this case some discriminant information is also lost [11].

3.2.3 Adaptations of Linear Discriminant Analysis

There exist other methods similar to LDA. The most known variations are Quadratic Discriminant Analysis (QDA) and Multivariate Analysis of Variance (MANOVA).

3.3 Comparison of the methods

In this section a comparison will be made between the two dimensionality reduction methods proposed above, PCA and LDA in Sections 3.1 and 3.2 respectively.

Both methods are used in this case as means of data preprocessing algorithms. Since the original data has a high dimensionality due to it being composed of various variables (Psuc, Tdis, etc.), it is therefore necessary to reduce its dimensions to be able to perform a subsequent classification in real-time. Both methods have certain similarities and differences that will be seen throughout this section, comparing them and showing an example of how each of them would act with the same dataset.

3.3.1 Similarities

Both methods are used in this case to reduce the dimensions of the original data for later classification.

Both LDA and PCA are linear transformation methods, that project the original data onto the eigenvectors, which are linear combinations of the main features. Both methods project/reduce the data to the desired number of dimensions (eigenvectors). These eigenvectors are the directions that collect the main properties, sorted from highest to lowest using the associated eigenvalues. Thereby reducing the data to the number of dimensions that define the original data in its entirety without a great loss of information.

3.3.2 Differences

As mentioned, both methods are used to reduce the dimensions of the original data, however, each method does it differently. PCA focuses on the attributes or directions that present the greatest variance of the data. While LDA tries to maximise the separation of the classes to which the data belongs to.

There is also a difference when training both methods. PCA is an unsupervised method requiring only the original dataset. Whereas LDA is a supervised method meaning that in addition to the original dataset it also requires the class labels for the corresponding samples.

3.3.3 Example

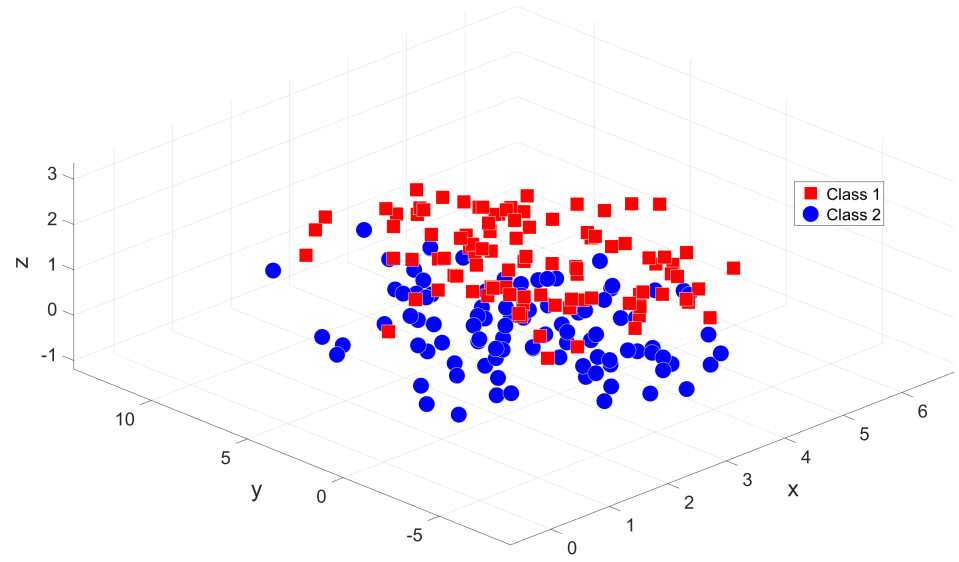
In this example, a series of points have been generated randomly using a normal distribution with a different mean and variance along each of the three original axes (x, y *and* z) as seen in Figure 3.5. These points belong to two different classes, represented by red squares and blue circles. The parameters for the generated data are specifically chosen to showcase the differences between both methods when reducing their dimensions. In this case, both data clouds/classes have the same mean in two of the three axes (x *and* y). However the variance differs in all of the axes for both classes with (y) having the highest variance compared to (x) having less and (z) with the least variance.

As seen in the figures, this distribution of the classes allows for obtaining clear directions for both the PCA and LDA methods, both main directions being different.

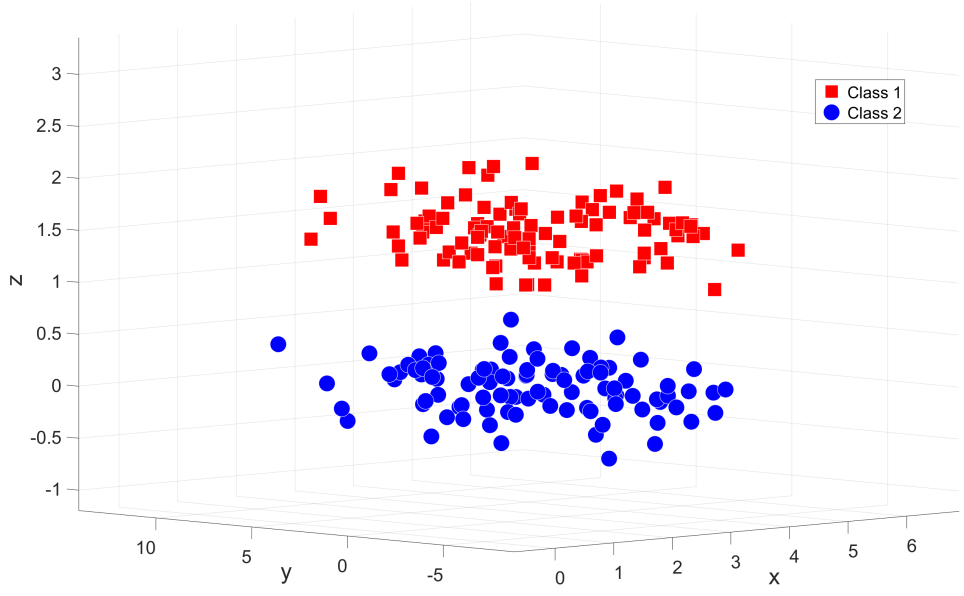
For the PCA method, both data clouds are considered as a single dataset, for this reason a clear direction of variance has been chosen along the y axis, this being the first Principal Component or PC1. The second PC corresponds to the second direction with the greatest variance, in this case x, compared to z (since both classes have been placed close enough). The results from this method can be seen in Figure 3.6, which shows a reduction from three dimensions down to two dimensions. Where the data is dispersed as much as possible, regardless of whether the two classes are intermingled.

The LDA method looks for those directions in which the classes are as far apart from each other as possible, while also clustering the observations within the classes. Similarly to PCA, LD1 is the direction that maximises these two sought-after characteristics, while LD2 is the second direction. The results from this methods can be seen in Figure 3.7, where a clear separation between the two classes can be observed, reducing the dimensions of the original three to the two main LD's.

The conclusion of this example is that both methods are equally valid for reducing the dimensions, they only differ in the features that they seek to highlight.



(a) Top view of the generated data



(b) Side view of the generated data.

Figure 3.5: Data generated for comparison between PCA and LDA methods

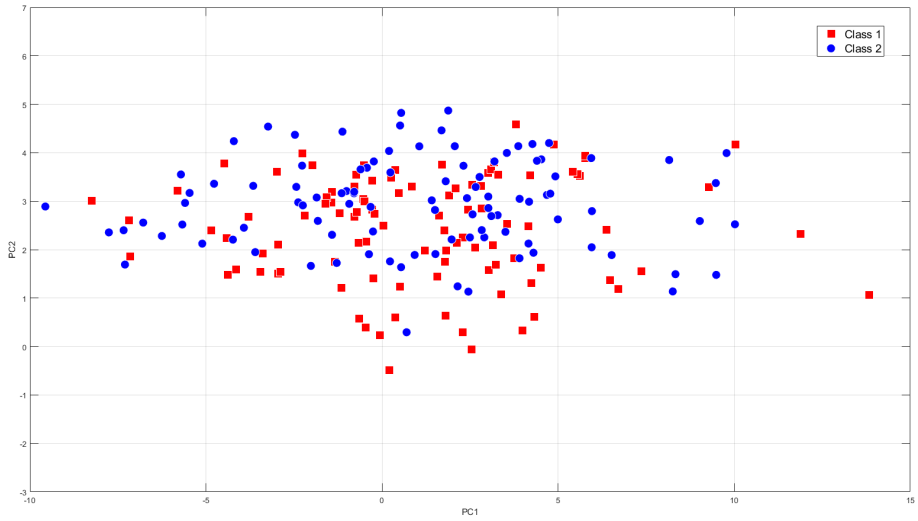


Figure 3.6: PCA method

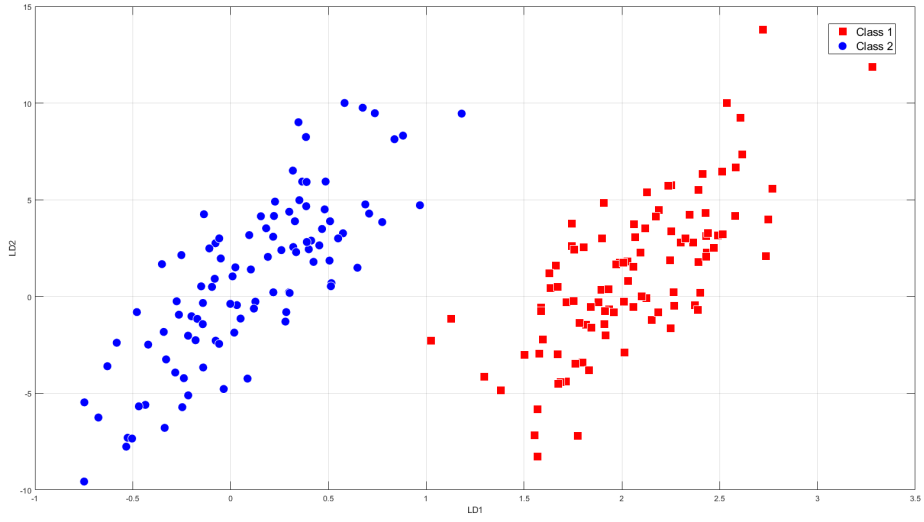


Figure 3.7: LDA method

3.4 Support Vector Machines

The method of classification using Support Vector Machines (SVM) was developed during the '90s. Originally, it was created as a binary classification method, i.e., classifying whether an element belongs to one of two classes. However, the method has later been extended to classification of a larger number of classes, as well as used in regression problems.

The SVM method is based on finding a hyperplane that can separate both classes with the fewest possible outliers. In a D-dimensional space, a hyperplane is defined as a flat and affine (it does not need to pass through the origin) subspace of $D-1$ dimensions. This means that in a bi-dimensional space, the resulting hyperplane would be a line (1 dimension), while in a 3-dimensional space, the hyperplane would be a plane (2-dimensions), and in the same way for spaces with $D > 3$.

The mathematical definition of the hyperplane can be expressed with the following equation:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D = 0 \tag{3.16}$$

Where D is the dimension of the space, and given the parameters w_0, w_1,..., w_D, the point defined by $x = (x_1, x_2, \ldots, x_D)$ that fulfils Equation 3.16 belongs to the hyperplane. When point x does not satisfy Equation 3.16, it means that it is on one side or the other in the regions separated by the hyperplane as seen in Figure 3.8. To know which side of the subdivided regions the point falls, only the sign of the equation needs to be computed.

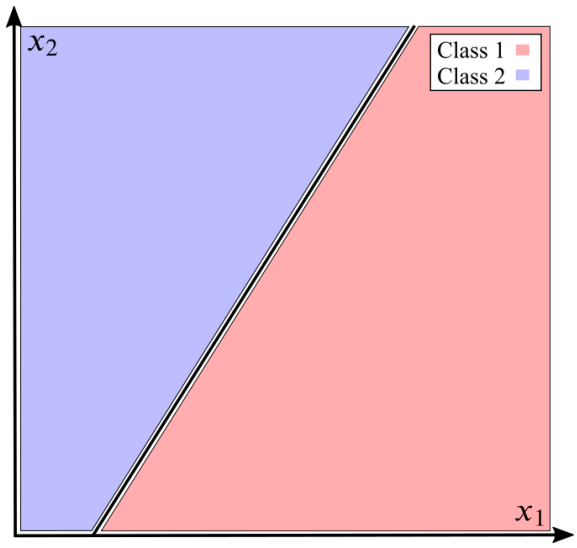


Figure 3.8: SVM hyperplane separating the space into two classes

There are two possibilities when finding the hyperplane: cases in which data can be separated linearly, and cases in which it cannot be linearly separated.

3.4.1 Linear separable cases

In the case that the variables are perfectly separable into two classes (+1 and −1), the hyperplane separates the space according to the following inequalities:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D < 0 \tag{3.17}$$

When the point is labelled as −1 and

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D > 0 \tag{3.18}$$

when the point is labelled as +1.

Each new observation (x^*) can therefore be classified by observing the sign of the function $f(x^*) = w_0 + w_1 x_1^* + w_2 x_2^* + \cdots + w_D x_D^*$. Furthermore, the magnitude of the value obtained indicates the distance at which the observation is from the selected hyperplane. This function can be compressed in a matrix formulation as:

$$y(x) = w^T x + b \tag{3.19}$$

Where $w^T = (w_1, w_2, \ldots, w_D)$ is the weight vector, and $b = w_0$ is a bias.

However, the choice of the hyperplane can be complicated, since there are infinite hyperplanes that satisfy the inequalities, which is why some criteria need to be added to find the most optimal one.

The solution is known as the maximal margin hyperplane. This is the furthest hyperplane with respect to all training observations. To obtain it, the distances of each observation perpendicular to the hyperplane have to be calculated. Of these distances, the one with the smallest measure is called the margin, and shows how far apart the hyperplane is from the observations as seen in Figure 3.9. The distance from a sample point to the hyperplane is defined by:

$$\frac{t_n y(x_n)}{\|w\|} = \frac{t_n (w^T x_n + b)}{\|w\|} \tag{3.20}$$

Where t_n are the target values or labels (in this case $+1$ and -1). The maximal margin hyperplane is the one with the largest margin of all the hyperplanes. Even so, the problem is not easy to solve, since there is still an infinite number of hyperplanes by which to classify the measurements, so optimisation methods are used. Thus, to find the maximal distance from the closest sample to the hyperplane the following equation needs to be solved:

$$\underset{w,b}{\operatorname{argmax}} \left\{ \frac{1}{\|w\|} \min_{n} [t_n(w^T x_n + b)] \right\} \tag{3.21}$$

In order to solve this optimisation problem the following Lagrangian function is used:

$$L(w, b, a) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(w^T x_n + b) - 1 \right\} \tag{3.22}$$

where, L is the Lagrangian and a is the Lagrangian multipliers. The Lagrangian function minimises with respect to w and b and maximises with respect to a. By taking the derivative of $L(w, b, a)$ with respect to w and b equal to zero the following equation is obtained.

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \tag{3.23}$$

where, $\phi(x_n)$ is the transformation of x. Once the weight vector is obtained the next step is to determine the bias parameter b.

$$b = \frac{1}{N_{Sv}} \sum_{n=1}^{Sv} (t_n - \sum_{m=1}^{Sv} a_m t_m k(x_n, x_m)) \tag{3.24}$$

where, N_{Sv} is the total number of support vectors and $k(x_n, x_m)$ is the kernel function. However, if the kernel function is not implemented it can be interpreted as $\phi(x_n)$.

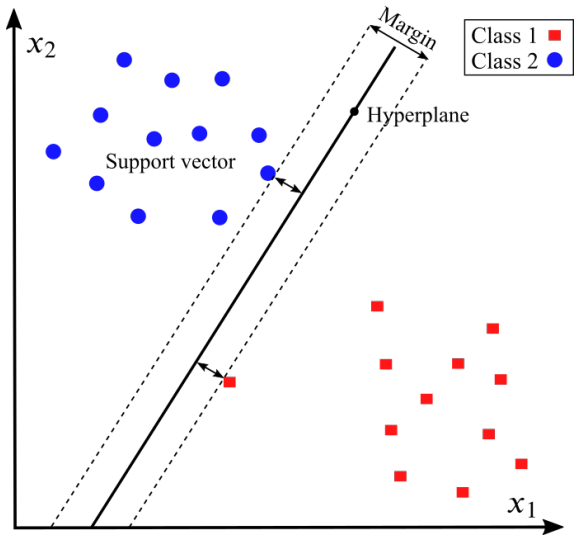


Figure 3.9: Visualisation of how the maximal margin hyperplane classifies the data

The observations that are on the edge of the margins are called support vectors, and they support (determine) the maximal margin hyperplane. Any modification to these observations would imply a drastic change in the maximal margin hyperplane, unlike the rest of the observations that do not affect the hyperplane when modified.

3.4.2 Non-linear separable cases

The case described in the previous subsection is an ideal and simple example of data separation using a hyperplane. However, it is not always the case that the data can be linearly separated using a hyperplane that perfectly differentiates one class from another. In some cases the data can be found intermingled causing it to not have a maximal margin hyperplane. In these cases modifications to the concept are accepted in which some hyperplanes that separate most of the data can be considered valid, allowing a small number of outliers. This type of hyperplane is known as Support Vector Classifier or Soft Margin SVM.

Even on the occasions when a maximal margin hyperplane may exist, it may present certain drawbacks that make it sub-optimal:

- Since the hyperplane has to separate the two classes perfectly, it is very sensitive to any small changes in the data. Adding new observations can drastically change the hyperplane, so the method has little robustness.
- The fact that it has to fit perfectly to all the observation can result in overfit-

ting for the training data, making the classification of new data biased to the training data.

For this reason, the use of Support Vector Classifiers is sometimes a better option. These, sacrifice not finding the maximal margin, allowing some observations to be found within them or even misclassified as seen in Figure 3.10. With this, a more robust classifier is achieved with a greater predictive capacity against new observations, solving the problem with overfitting.

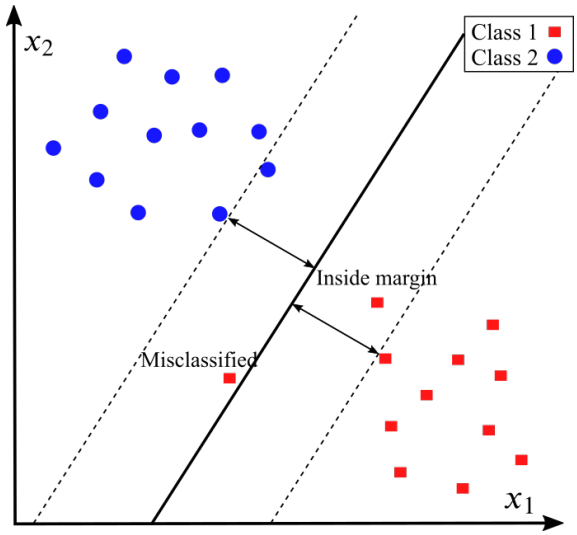


Figure 3.10: Visualisation of how the soft margin hyperplane classifies the data

Solving the classification problem by using a soft margin SVM is a convex optimisation problem in which the regularisation parameter C comes into play. This parameter controls how restrictive the margins will be, allowing a greater or lesser amount of errors and tolerances. Minimising the following equation would be equivalent to maximising the margin width.

$$C\sum_{n=1}^{N}\xi_n + \frac{1}{2}\|w\|^2 \tag{3.25}$$

where, w is a weight factor and ξ_n is a slack variable used for penalising the points which are misclassified. When the data points are correctly classified or on the margin line, $\xi_n = 0$. However if they lie on the correct side but inside the margin, $\xi_n < 1$. Lastly if they are on the wrong side of the hyperplane as misclassified points $\xi_n > 1$.

If $C = \infty$, no error is allowed, therefore it will be equivalent to a maximal margin hyperplane. The closer C is to zero, the less the outliers are penalised and hence the less accurate the margin will be. Consequently, C is the parameter in charge of balancing the bias and the variance of the model. Similar to the problem when the data is perfectly linearly separable, the data that are on the edge of the margin are those that influence the selected hyperplane, however, in this case those observations that fall within the margin itself also influence it. The lower the value of C, the greater the width of the margin, therefore it will cover a greater number of observations (both at the edge and within it) and it will be more robust, hence increasing the bias, but decreasing the variance. On the contrary, when the value of C increases, the width of the margin decreases, covering a smaller number of observations (previously called support vectors) and the classifier will have a lower bias but a greater variance.

After introducing both the ξ and C variables the Lagrangian optimisation problem Equation 3.22 changes to the following equation.

$$L(w, b, a) = \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n \sum_{n=1}^{N} a_n \{t_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n \tag{3.26}$$

The fact that only a few observations (support vectors) influence the selected hyperplane makes this method more robust. As it does not depend on all the observations when determining the hyperplane.

3.4.3 SVM

The Support Vector Classifier described above has good results when the separation between the classes is more or less linear, decreasing its usefulness when this is not the case. However, the fact that the classes are not linearly separable in the original space does not mean that they can not be in spaces with higher dimensions.

The SVM method can be considered as an extension of the Support Vector Classifier by increasing the size of the data to obtain a linear separation. This separation can be achieved through the use of a Kernel.

3.4.3.1 Kernel

The dimension of a dataset can be modified by increasing or combining any of its dimensions. There are infinite transformations, so it would be very difficult to choose the one that best suits each occasion which is why the kernels are used for this purpose. The kernel is a function that returns the dot product between two vectors of the original space in a new space with a greater dimension. This step is commonly called the "kernel trick", since with a small modification of the original problem, a solution can be obtained in any dimension. There are many kernels, but the most known are:

- **Linear kernel**: $K(x, x') = x \cdot x'$

  Where K is the kernel function, x and x' are both different samples of the data set. If a linear kernel is used, the result will be equivalent to a Support Vector Classifier.

- **Polynomial kernel**: $K(x, x') = (x \cdot x' + r)^d$

  Where d is the polynomial degree and r is the polynomial coefficient. When $d = 1$ and $r = 0$, the result is the same as the linear kernel. If $d > 1$ the limits of the margins will be non-linear, increasing this non-linearity as d increases. These parameters are determined by cross-validation methods.

- **Gaussian kernel (RBF)**: $K(x, x') = \exp(-\gamma||x \cdot x'||^2)$

  In this kernel, γ is the parameter that controls the behaviour of the kernel. With small values, the kernel resembles a linear kernel, while increasing its value, the kernel becomes more flexible, while its non-linearity increases.

3.4.4 SVM for more than two classes

As mentioned at the beginning of the section, SVM is an extended method of binary classification, classifying between two classes, so it is not designed to classify more than two classes. However, different methodologies allow the classification among a greater number of classes. Among the most known are: one-versus-one, one-versus-all and DAGSVM.

- **one-versus-one**: Assuming a scenario in which there are c ($c > 2$) classes and it is desired to classify a new observation using SVM. The procedure would be to generate $c(c-1)/2$ SVMs comparing all possible combinations of classes. In order to classify the new observation, the $c(c-1)/2$ SVMs would have to be used and at the end, the class that would have appeared with the greatest

frequency in each of the binary classifications would be the one assigned to the new observation. This methods disadvantage is that the number of models required increases greatly as classes increase.

- **one-versus-all**: This method consists of training c different SVMs comparing each of the classes against the remaining $c-1$ classes. To obtain a prediction, each of the SVMs is used, and the class that yields a positive outcome will be assigned to the observation. This method has the drawback that several classes can result successfully at the same time. In addition, this method is unbalanced when training, since there are many more negative than positive samples in each SVM.
- **DAGSVM**: The Directed Acyclic Graph SVM method is an improvement of the one-versus-one method. The procedure is the same, except that a reduction in execution time is achieved. The method begins by making a binary comparison, and once classified as one of the participating classes (suppose A versus B), all subsequent comparisons with the unselected classes are discarded (in the case of having classified as A, the comparisons of C vs. B, D vs. B, etc. are discarded). This greatly reduces the number of total comparisons required.

CHAPTER 4

Implementation

In this chapter brief descriptions of the implementation of each developed method (PCA, LDA and SVM) will be done, as well as the combination of these in order to classify new data for detecting faults in the system.

4.1 PCA

The PCA method has been implemented as a MATLAB function, so it can be used in a more accessible and flexible way.

The function receives as inputs a matrix with all the samples (in which one class is concatenated with the other) and the number of Principal Components to which the data is to be projected (nPC). The matrix with the samples has dimensions of $D \times n$, where n is the number of samples and D is the number of different signals in the matrix (T_{suc}, T_{ret}, P_{dis}, etc.).

The code is based on the equations developed in Section 3.1, and follows the steps described. First, the covariance matrix S is calculated according to Equation 3.3. Second, the eigenvectors as well as the eigenvalues of S are computed using the MATLAB function "$eig(S)$", as described in the eigenvector problem in Equation 3.4. The third step consists of sorting these eigenvectors from highest to lowest using the associated eigenvalues. The last step is to create the transformation or projection matrix U, of dimensions $D \times nPC$.

As outputs of the function, the transformation matrix is returned, along with the sorted eigenvalues and eigenvectors.

4.2 LDA

The LDA method has also been implemented as a MATLAB function, to make it easier to access and obtain the desired projection matrix.

The function receives as input parameters the $D \times n$ matrices of the two classes (n being the number of samples, and D the number of variables/dimensions, in this case 13), as well as the number of dimensions the data is desired to be reduced to (nLD). As output parameters, the function returns the projection matrix W ($D \times nLD$) from Equation 3.15, the matrices with the calculated eigenvectors and eigenvalues and a vector with the labels of each sample showing which class they belong to (-1 and $+1$) needed for the SVM classifier.

The code is based on the equations developed in Section 3.2, and can be seen in Appendix A.2. The between- and within-class covariance matrices S_B and S_W are calculated using Equations 3.7 and 3.9 respectively and finally the eigenvalues and eigenvectors of $J(w)$ (Equation 3.10) are computed using the MATLAB function "$eig(S_B, S_W)$", which is equivalent to solving Equation 3.11. The eigenvalues are then sorted in descending order along with their corresponding eigenvectors. Finally, the transformation matrix W is created with the desired number of eigenvectors which will determine the new dimensions of the transformed data.

4.3 SVM

The SVM classifier has been implemented as a MATLAB function in order to simplify the process of creating new models using the projected data from the dimensionality reduction methods.

The SVM function receives as inputs the training and validation data which are split from the projected dataset. It also gets a vector of possible regularisation parameters (C) used for finding the optimal parameter for the specific dataset. As output it returns the weight vector (w), bias (b) and the support vectors (Sv) which are all necessary for constructing the hyperplane, margins and predicting the classes of the test data.

The first step is to write all the necessary terms for utilising MATLAB's "quadprog()" function which simplifies the process of solving the quadratic problem. In this case it concerns solving for a (Lagrangian multipliers) which are used to determine the support vectors, weight vector and bias (Equation 3.22). The next step is to calculate the weight vector according to Equation 3.23 and the bias parameter

following Equation 3.24. Once these terms have been determined, the classes of the validation data can be predicted. The prediction procedure solves Equation 3.19 as the sign of it determines which class the data belongs to. These steps are run in a loop for all the possible C parameters and the accuracy of the predictions are stored to determine the optimal regularisation parameter (C). The SVM procedure is then rerun using the optimal regularisation parameter to get the output with the highest accuracy.

4.4 Classifiers combination

In order to achieve a fast classifier that can run in real-time, it is convenient to work with data in a lower dimensional space. Since the original data is collected in $D = 13$ dimensions, it is required to perform a dimensionality reduction prior to SVM classification. Figure 4.1 shows a flowchart of the classification procedure, starting from the extraction of the data to the prediction of new samples. The different options proposed in the chart will be described in the following sections.

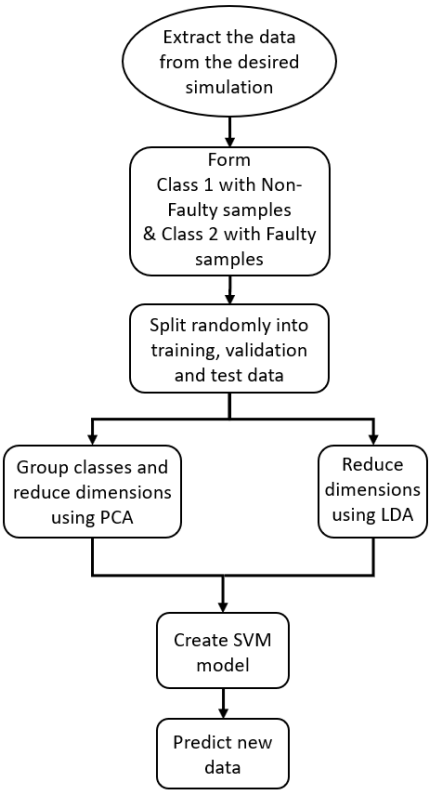


Figure 4.1: Flowchart of the PCA/LDA SVM implementation

4.4.1 PCA SVM

In this section the implementation of the combined PCA and SVM for classifying new data will be explained.

As seen in Figure 4.1, the first step is to extract the signals from the desired simulation tests. These signals are grouped into two classes, Class 1 with the samples of the non-faulty segment and Class 2 with the samples of the faulty segment. These classes are $D \times n$ matrices in which the rows are formed by each of the chosen signals. In order to analyse the static part of the signal and avoid the dynamic part, both the first sample and the number of samples are specified by the user. Classes are in turn separated for training, validation and unbiased test data. The number of samples in each of these data groups are determined by the user (see Section 6.1), where all the samples inside them are randomly selected among both class matrices maintaining the row order.

For the dimensionality reduction by means of PCA, both training classes are grouped in a $D \times 2n_{training}$ matrix, which is fed to the developed function (see Appendix A.1) and returns the transformation matrix $D \times nPC$ that reduces the dimensions of the original data into nPC dimensions using Equation 3.6. Both the training data and the validation data are projected using this matrix and then introduced in the developed SVM function (see the code in Appendix A.3).

The parameters that this function returns (w, b and Sv) are used to generate the hyperplane that separates both classes, as well as the margins and to determine the support vectors.

In order to classify new data, the sign of Equation 3.19 needs to be computed. x in this case being the projected coordinates of the test data using the already calculated transformation matrix from the PCA function.

4.4.2 LDA SVM

Since the methodology to perform the combination of LDA and SVM is very similar to that explained in the previous Section 4.4.1. Only a brief explanation of the differences and similarities of applying LDA as a reduction method and SVM as a classifier will be described.

As mentioned in the previous section, the first step is to extract the data from the desired simulation test. However, for this methodology both classes will remain separate. The classes are then fed to the LDA function (see Appendix A.2),

which returns the transformation matrix W that reduces the classes into the desired number of dimensions nLD using Equation 3.15.

The procedure from this point is the same as in the previous section: train the SVM with the training and validation data and obtain the parameters that define the hyperplane and margins.

Finally, to analyse which of the classes a new sample belongs to, the sign of Equation 3.19 is computed with x being the coordinates of the sample being projected with W from LDA.

CHAPTER 5

Validation Tests

Different tests have been carried out to check the validity of the implemented methods. In this chapter each of the tests will be explained, with a brief description of how they were developed and the results obtained from them.

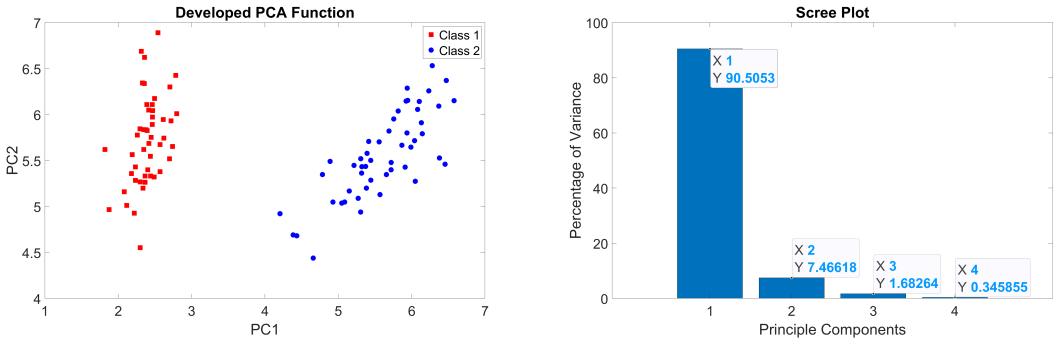
5.1 Validation 1: Verification of the methods

In order to verify the implementation of the methods used throughout this thesis. A couple of tests have been made comparing the developed methods to MATLAB's own functions for dimensionality reduction and classification. This is to ensure that the code developed based on the theory in Chapter 3 is functioning as intended.

5.1.1 PCA

The first test is a comparison between the developed function for PCA which can be seen in Appendix A.1 and MATLAB's *pca* function. The objective of this test is to illustrate the similarities between both functions by reducing the dimensions of a two class dataset with four dimensions down to two dimensions.

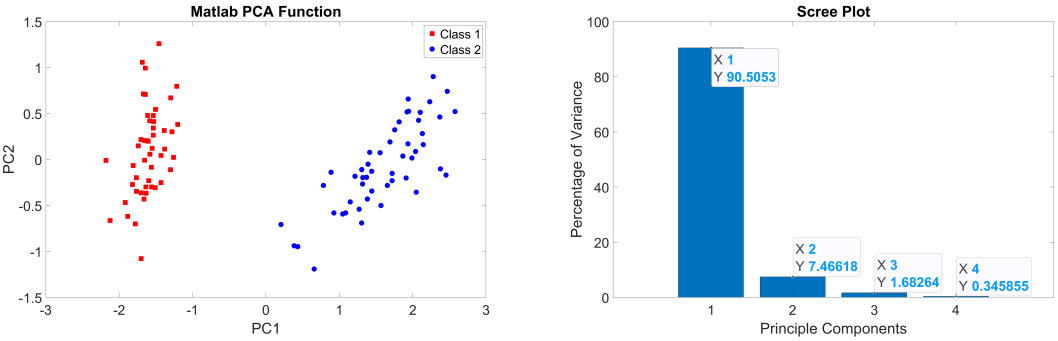
The results from the developed function for PCA can be seen in Figure 5.1. Where Figure 5.1a displays the projected data of the two largest Principle Components which collectively represents around 97 % of the information from the original four dimensional dataset as seen on the scree plot in Figure 5.1b.



(a) The projected data using PCA (b) Scree plot using PCA

Figure 5.1: The results from using the developed *PCA* function

The MATLAB function $pca(X)$ is simple to implement. As it takes the dataset as input and can simply output the transformed data, eigenvectors, eigenvalues and variance. This greatly simplifies the process of using PCA for dimensionality reduction and the results of this function can be seen in Figure 5.2. Similarly to the other figures both the projected data and scree plot are shown to compare the similarities between both functions.



(a) The projected data using PCA (b) Scree plot using PCA

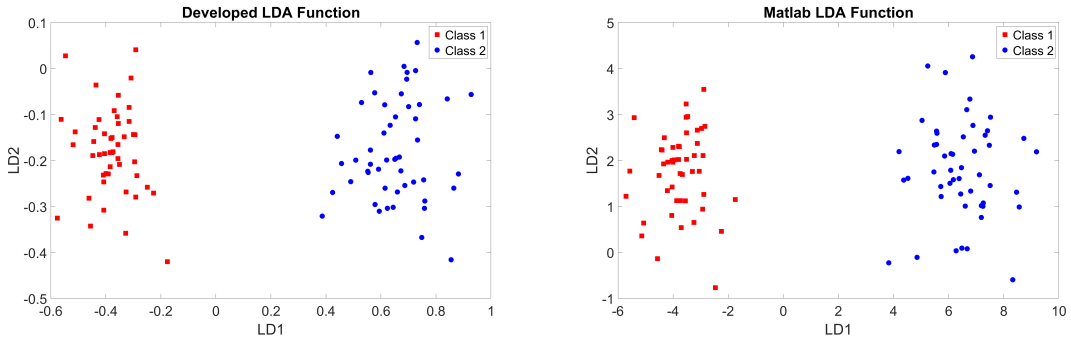
Figure 5.2: The results from using MATLAB's *pca* function

From both Figures 5.1 and 5.2 can it be concluded that the results of the projected data and scree plots are very similar. However with the exception that the projected data seems to be centred using the MATLAB function. This should however not pose a problem as the spread of the projected data is the same for both functions and the "percentage of variance" for both scree plots are identical to each other. This test therefore confirms that the developed function for PCA is working as intended.

5.1.2 LDA

The second test is a comparison between the developed function for LDA which can be seen in Appendix A.2 and MATLAB's function for LDA. The objective of this test is similar to the test for PCA as the goal is to illustrate the similarities between both functions by reducing the same two class dataset in four dimensions down to two using LDA instead.

The MATLAB function $fitcdiscr(X, Label)$ is simple to implement however it does not directly output the transformed data like in the case for PCA. Instead, it outputs a lot of different variables from which the most noteworthy for the case of dimensionality reduction are the S_B and S_W, which can be used to compute the transformation matrix using Equation 3.12 from Section 3.2. The original data can then be transformed using this matrix and plotted as seen in Figure 5.3b.



(a) The developed LDA function (b) MATLAB's $fitcdiscr$ function

Figure 5.3: Comparison between the developed LDA function and MATLAB´s $fitcdiscr$ function

The results from both functions for the two class dataset can be seen in Figure 5.3. Clearly illustrating that both figures are not similar to each other. However, they both seem to split the data in a similar manner. This is most likely due to the fact that the MATLAB function is made for multiple classes ($K > 2$) which the developed function is not, as it is not necessary for the purpose of this thesis which is to classify a two class dataset (Non-faulty and Faulty data). Meaning that the calculations used for the MATLAB function are not completely similar to the developed function. However, if the code is rewritten to accommodate multiple classes instead of only two, the figures for the projected data will most likely look very similar to each other.

To accommodate the multiple class LDA, some of the equations in Section 3.2 need to be rewritten as they differ slightly from binary class LDA. First and foremost the

calculation for S_B differs from Equation 3.7 in the following way.

$$S_B = \sum_{k=1}^{K} N_k(\mu_k - \mu)(\mu_k - \mu)^T \tag{5.1}$$

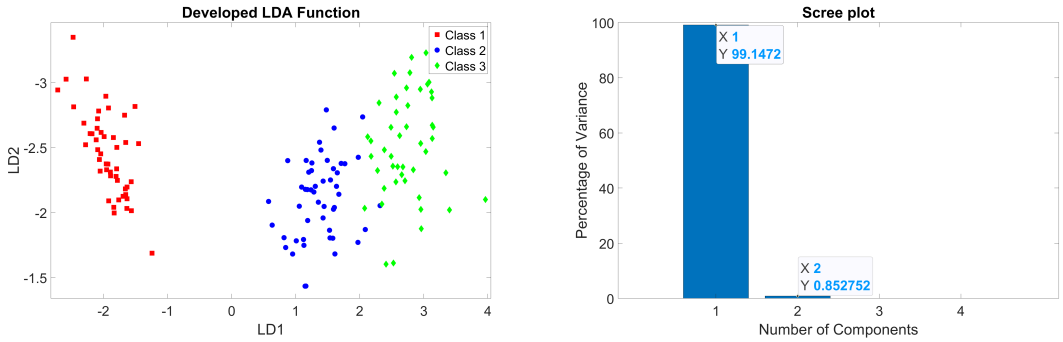
Where, K is the number of classes and N_k is the number of samples in each of the classes. Whereas μ_k is the mean of each class, μ is the mean of the total dataset with N being the number of total samples in the dataset.

$$\mu_k = \frac{1}{N_k} \sum_{k=1}^{N_k} x_n, \qquad \mu = \frac{1}{N} \sum_{k=1}^{K} N_k \mu_k \tag{5.2}$$

The other equation which differs is the S_W which is rewritten from Equation 3.9 to the following equation:

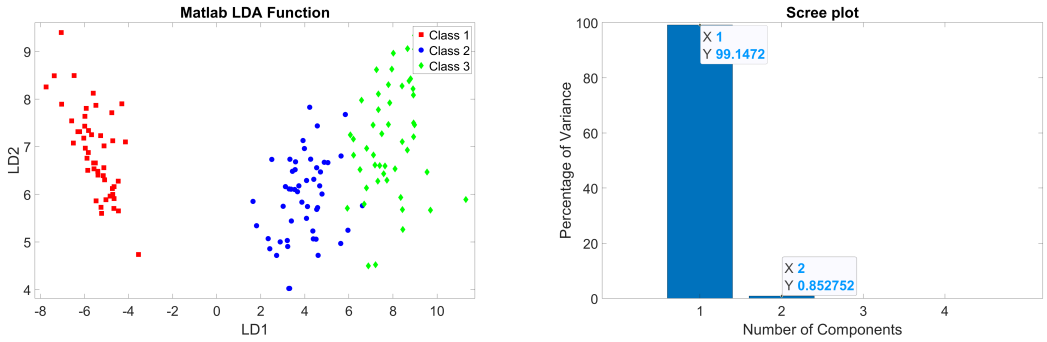
$$S_W = \sum_{k=1}^{K} \sum_{n=1}^{N_k} (x_n - \mu_k)(x_n - \mu_k)^T \tag{5.3}$$

Besides these two equations the rest should be similar to the ones mentioned in Section 3.2. The results from this rewritten implementation compared to MATLAB's function can be seen in Figures 5.4 and 5.5 for a three class four dimensional dataset reduced down to two dimensions.



(a) The projected data using LDA (b) Scree plot using LDA

Figure 5.4: The results from using the developed LDA function for multiple classes



(a) The projected data using LDA (b) Scree plot using LDA

Figure 5.5: The results from using MATLAB's $fitcdiscr$ function for multiple classes

It can be concluded from Figures 5.4 and 5.5 that the reason for the difference in the plotted diagrams in Figure 5.3 is because of the S_B and S_W equations being calculated using the general mean. This is due to MATLAB's function being made for multiple classes which is demonstrated when the developed function in Appendix A.2 is rewritten for multiple classes. As it gives a very similar result as the MATLAB function where the scaling in the x and y-axis is the only difference. This is further supported by the fact that the scree plots are identical to each other. It is therefore not possible to compare the MATLAB function to the developed two class LDA function, however it can be assumed that it is working as intended. Since the results from both multiple class dimensionality reduction functions seem to be similar.

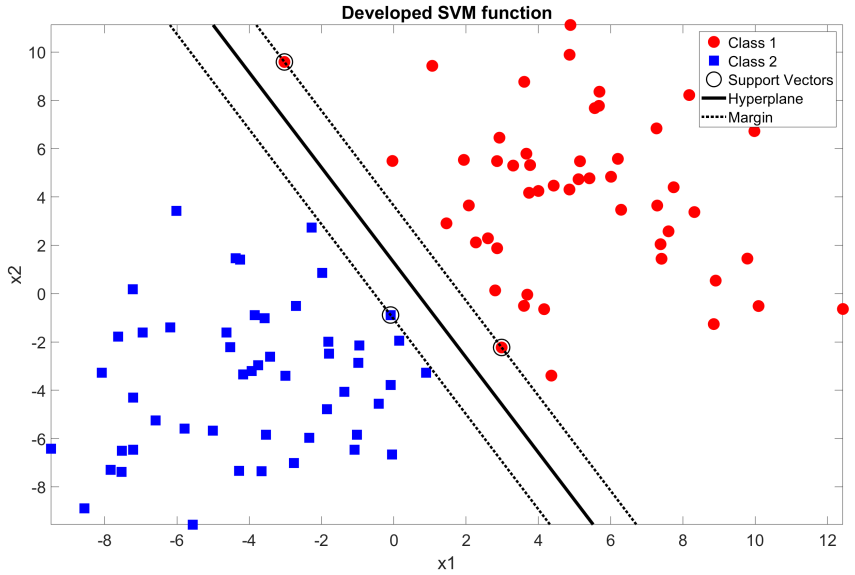
5.1.3 SVM

The third test is a comparison between the developed SVM function which can be seen in Appendix A.3 and MATLAB's SVM function. The objective of this test is to find the optimal hyperplane and margins which split the two class dataset into two regions. By comparing the results from both functions it should verify whether the developed SVM function is working as intended.

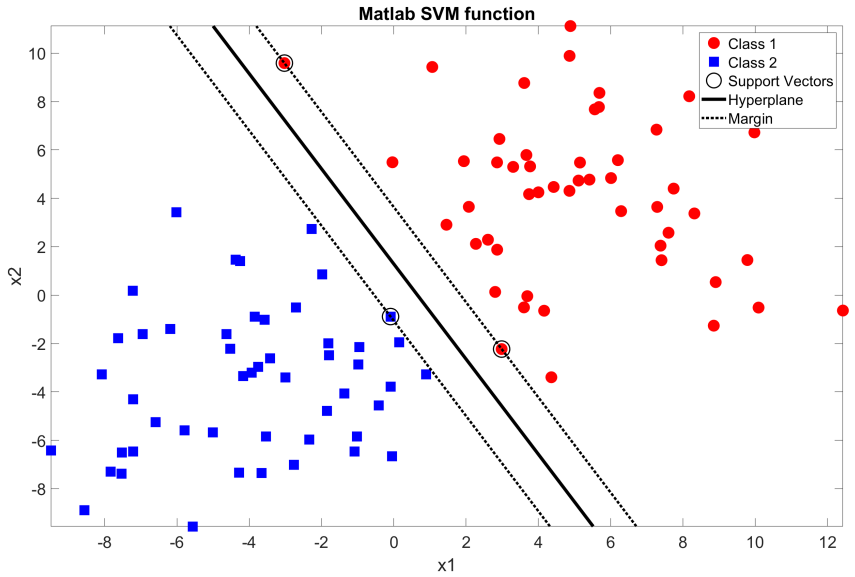
The MATLAB function $fitcsvm(X, label)$ is simple to implement as it only requires the data of a two class dataset (grouped in X) and the corresponding labels as input and outputs the necessary variables to create the SVM. Some of these variables include a, w, b, S_v and more. This simplifies the process of creating the hyperplane, margins and determining where the support vectors lie.

Figure 5.6 illustrates the similarities between both functions for SVM. The results from the figures show that the hyperplane, margins and support vectors are all

placed in the same position for both functions. It can therefore be concluded from this test that the developed SVM function seems to be working as intended.



(a) The SVM results using the developed method



(b) The SVM results using MATLAB's implementation

Figure 5.6: Comparison between the developed SVM function and MATLAB's SVM function

5.2 Validation 2: Dimensionality reduction methods

To verify that the dimensionality reduction is done correctly for both PCA and LDA, it has been tested that the developed transformation matrices U and W work with validation data.

For this test, only one of the multiple datasets mentioned in Section 2.3 has been used as training data. The 13 different signals have been extracted and grouped into two classes, C_1 containing $N = 2000$ non-faulty samples, and C_2 with 2000 faulty samples. Each of the signals make up a row in the class matrix, where the columns are the samples. The $D \times N$ matrices formed are then grouped into a new matrix called C_{all}. This is needed since each of the methods treats the data differently. PCA processes all the data in the same way, since it is an unsupervised method, while LDA needs to know the labels of the samples.

As seen in Figure 5.7, the top sub figure shows the transformed original data using the U matrix from PCA to maximise the variance of the data along the main two Principal Components. Whereas the bottom sub figure shows the transformed original data using the W matrix from LDA to maximise the separability of both classes along the main two eigenvectors.

New data can then be transformed into these two main feature directions using the transformation matrices U and W obtained from the training data mentioned before. However to verify that both methods are working as intended, the validation data used for testing are from the same dataset but a different segment of the signals. Figure 5.8 shows the projection of the samples using the transformation matrices obtained from the training data in Figure 5.7 for both methods, the top sub-figure being PCA and the bottom sub-figure being LDA.

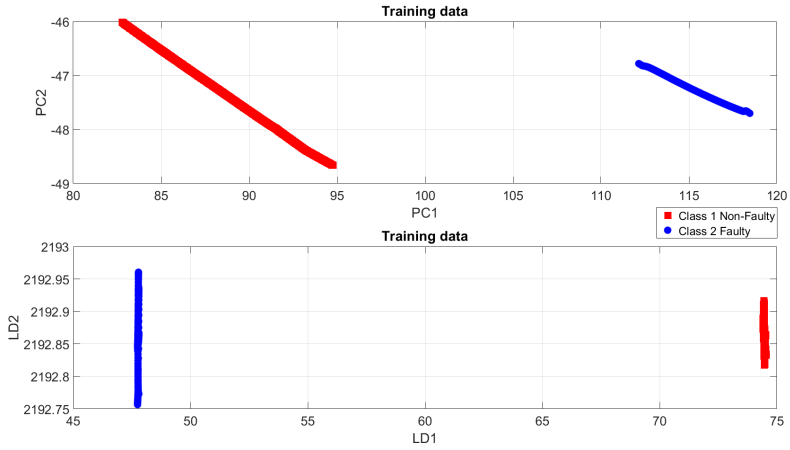


Figure 5.7: PCA and LDA methods using training data from one of the tests provided by Bitzer Electronics

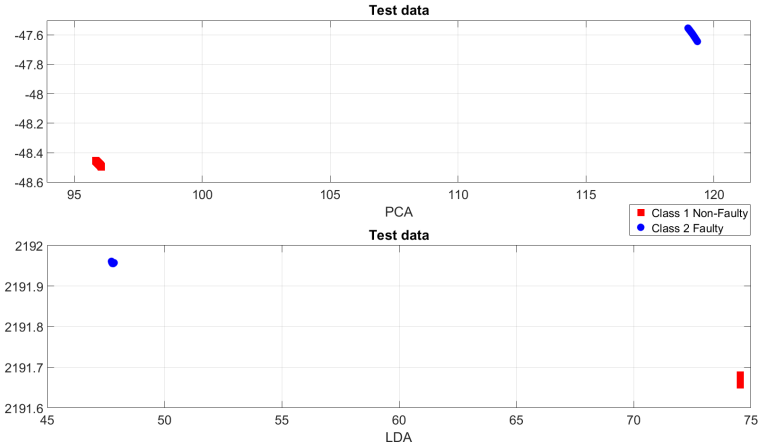


Figure 5.8: PCA and LDA using the obtained transformation matrices to project the test data

CHAPTER 6

Tests

In this chapter, there will be a description of the different tests carried out to evaluate the developed methods under different initial conditions. All tests will be performed following the same pattern to maintain consistency. The first test will be explained in more detail compared to the rest since they will keep the same parameters. The changes inherent to each test will be explained in its corresponding section.

6.1 Test 1: Classification of an easy dataset

The intended outcome of this test is to create a classifier that allows for classifying faulty samples that belong to signals that are easily distinguishable from the non-faulty samples. In this case the test has been done by choosing simulation 30 from the Table in Appendix B.1 where the faulty viable is P_{dis}, adding an offset of -1 bar.

The original data of this simulation can be seen in Figure 6.1. From this data, the two classes are extracted forming two matrices of $D \times nSamples$, where the $nSamples$ is 2000 and D is 13. To avoid the dynamic part of the signals, the first sample starts at sample 1000. The two class matrices are each randomly split into three matrices one for training, validation and unsupervised testing. 20% of the original matrix has been selected for validation while 10% is used for testing. This is equivalent to 400 and 200 samples respectively, whereas the rest of the samples are used for training.

The training data is then introduced to both dimensionality reduction methods. Where the data is projected from the original 13 dimensions down to only 2, which is sufficient (without a great loss of information) and allow for representing the data in a 2D graph. The training and validation data are then fed to the SVM, which

creates the optimal model using the chosen values of the regularisation parameter (C) between: [0.1, 1, 5, 10, 20, 50, 100]. Once the model is created, it can be tested using the previously projected test data.

The results of this test are shown in Section 7.1

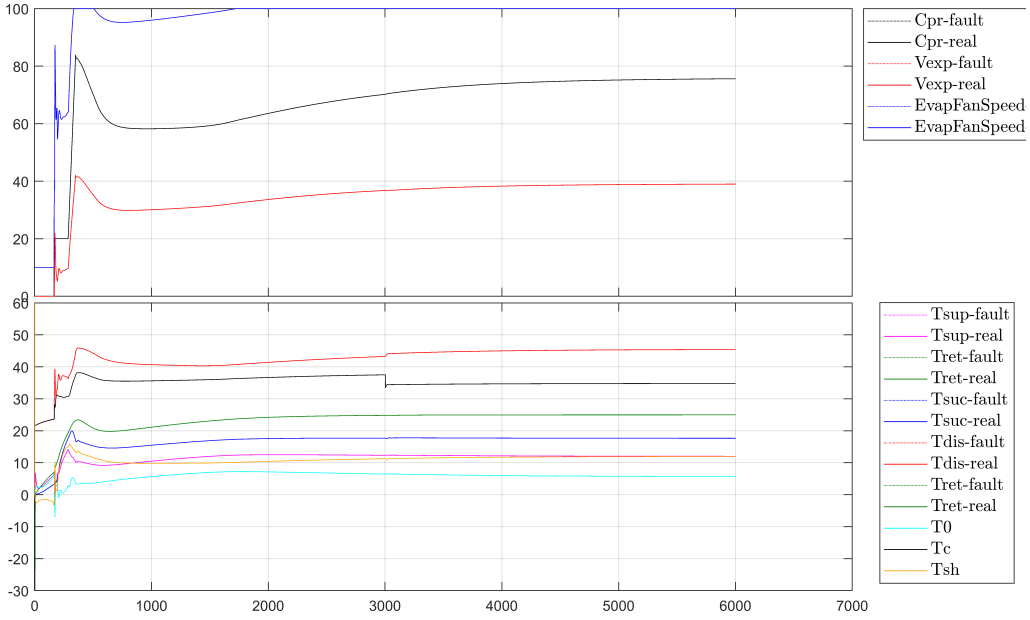


Figure 6.1: The original data from simulation 30

6.2 Test 2: Classification of a difficult dataset

The objective of this test is to check the performance of the classifier when the faulty samples are very similar to the non-faulty. For this, a simulation with low HeatLoad value has been chosen, which causes the signals to oscillate making the non-faulty and faulty samples overlap. In this case, simulation 21 has been chosen from Table B.1, and the variable that causes the failure is T_{dis} with an offset of $-2[°C]$.

The original data can be seen in Figure 6.2, where it can be observed that the non-faulty data and faulty data are very similar as they overlap each other.

The rest of the parameters remain the same as those mentioned for Test 1: (first sample, total number of samples, percentages into which the data is divided, etc.).

The results of this test are shown in Section 7.2

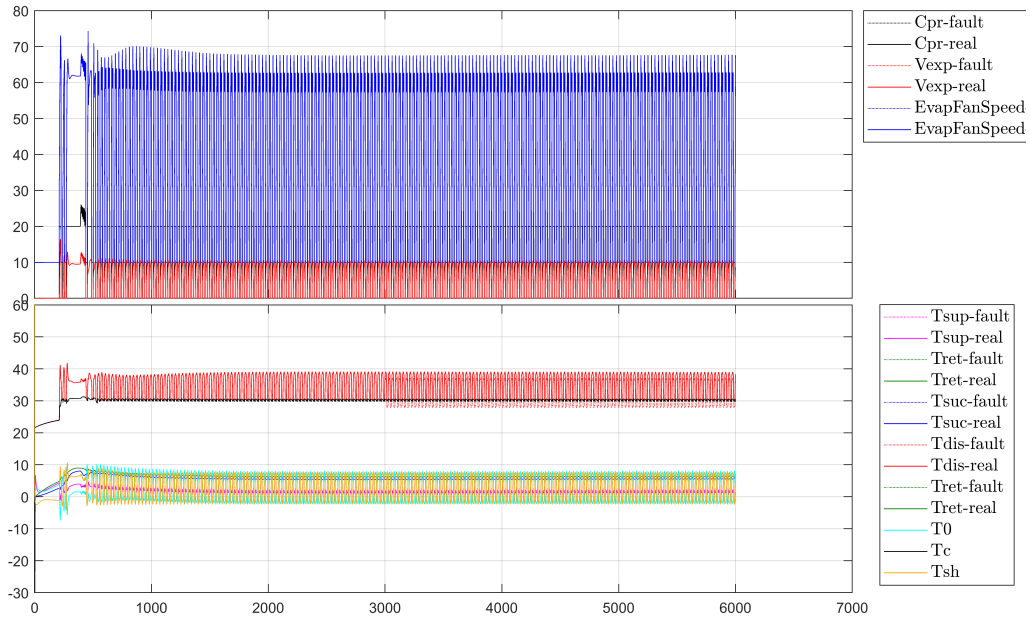


Figure 6.2: The original data from simulation 21

6.3 Test 3: Effect of the noise in the data

The purpose of the third test is to analyse the effect of noise on the signals as it is a common occurrence on real systems. Data is usually obtained with noise, which could for instance be caused by imperfect sensors.

In order to compare the results, the data is extracted from the same simulations as in the previous tests.

The effect of noise has been tested in different ways. First, noise has been applied to the training data, using the same procedures described in Sections 4.4.1 and 4.4.2. Secondly, noise has been applied to the testing data, which reproduces a real system. As a third test noise is added to both the training and test data to increase the prediction performance for more realistic data. Lastly, it has been applied to the whole dataset, including the validation data.

The noise is added as a random number for each sample with $\mathcal{N}(0, 2)$. This adds a deviation from the original data of [-2, 2], which for some variables is the value assigned for the offsets to simulate the faulty state. Making it harder for the SVM to separate the data as the Non-Faulty and Faulty data overlap each other. In [2]

noise of the same normal distribution was applied and it was said to be as high or even higher than what a real system could experience.

As an example, this noise can be seen applied to the T_{ret} variable in Figure 6.3.

Lastly, to analyse the robustness of the models against noise, the value of the variance applied to the noise has been increased and the effects on the accuracy of the predictions have been observed.

The results of this test are shown in Section 7.3.

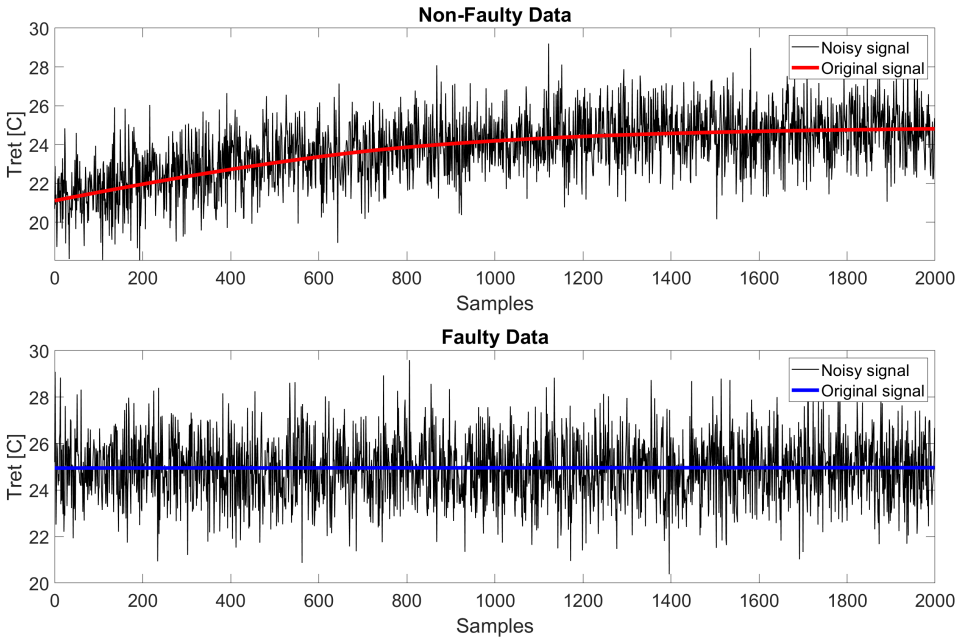


Figure 6.3: Noise applied to T_{ret} for non-faulty and faulty classes

6.4 Test 4: Using a different simulation as testing data

The fourth test focuses on evaluating how the proposed methodologies behave when test data is used from a different simulation. The objective of this test is not to create a general model that works for all types of failures, but rather see how it behaves with data from both similar and different simulations.

Keeping the training and validation data from simulation 24 in Table B.1, different

tests have been carried out modifying the origin of the testing data.

In the first test, the testing data was extracted from simulation 22 (see Table B.1) which has the same failure signal, T_{dis}, but slightly different $HeatLoad$ at 17000 instead of 20000 and T_{set} at 7 compared to 12.

Whereas in the second test, simulation 36 from Table B.1 was selected as it has the same $HeatLoad$ at 20000 and T_{set} at 12 but a different fault signal, P_{suc}.

In order to confirm the results another simulation (38) with a compressor fault having completely different $HeatLoad$ (13000) and T_{set} (0.0) parameters is tested.

To check the results with a simulation considered as difficult to separate, the same procedures previously described have been repeated for simulations with low values of $HeatLoad$.

The original signals used for training and validation data can be seen in Figure 6.4, while the signals used for the first test data can be seen in Figure 6.5 and the signals used for the second test data can be seen in Figure 6.6.

The results of this test will be described in Section 7.4.

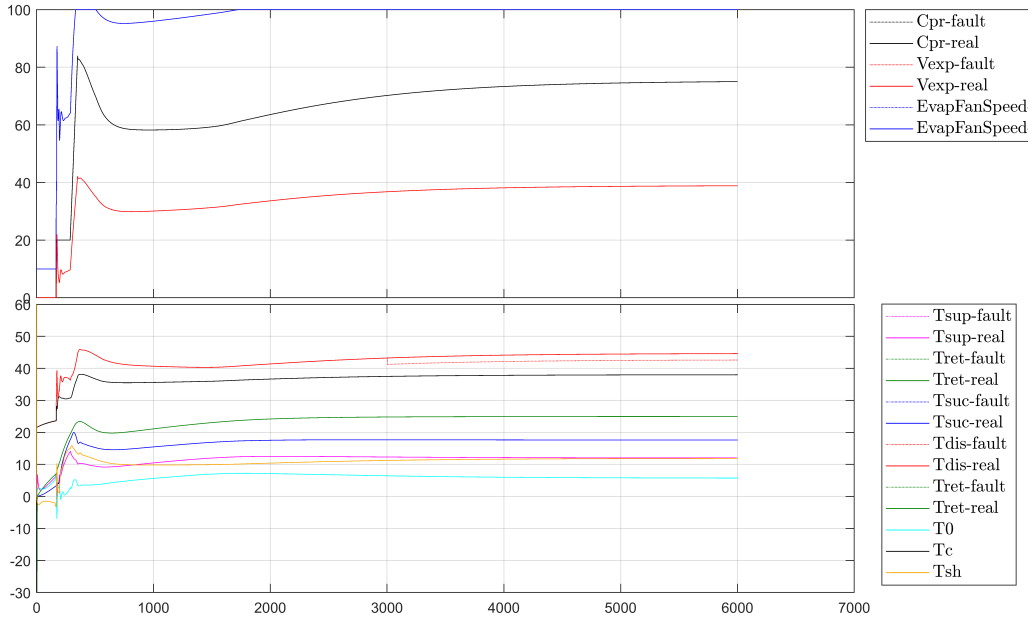


Figure 6.4: The original data from simulation 24 used for training and validation data

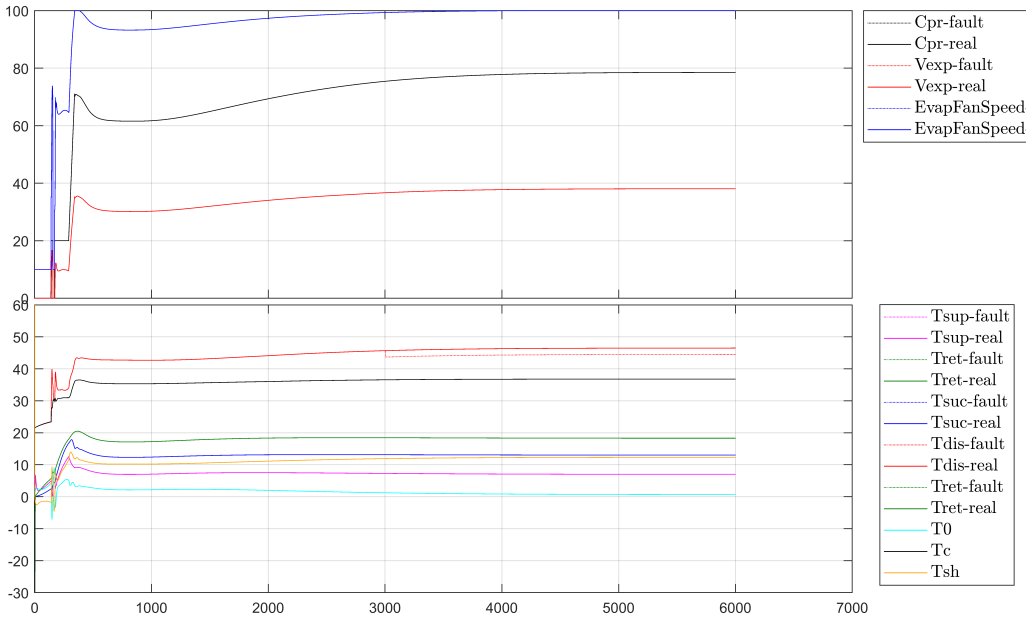


Figure 6.5: The original data from simulation 22 with the same faulty variable but different $HeatLoad$ and T_{set} parameters

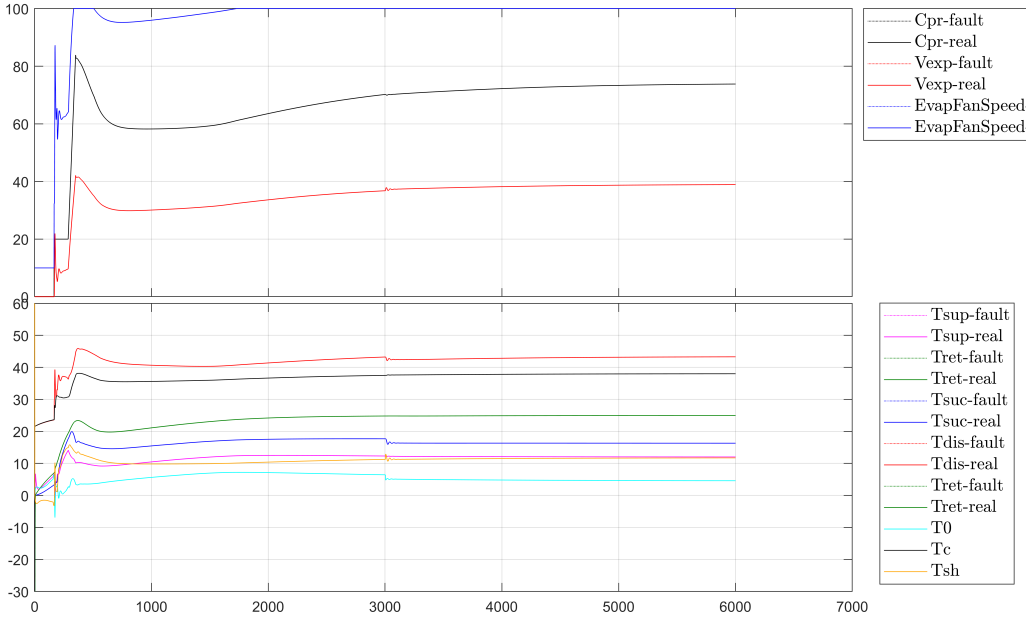


Figure 6.6: The original data from simulation 36 with the same $HeatLoad$ and T_{set} parameters, but different faulty variable P_{suc}

6.5 Test 5: Effect of the dynamics in the signals

In this test it is desired to evaluate the effect of the dynamic part of the signals in the developed classifiers. In order to contrast the results, the same simulations as for Tests 6.1 and 6.2 have been used for extracting the data (see Figures 6.1 and 6.2), but in this case the first sample begins at sample 50.

The results of this test will be shown in Section 7.5.

6.6 Test 6: Effect of standardising the data

As mentioned in the first step in Section 3.1.1, for some data it is necessary to standardise it before making the dimensionality reduction method. Not all the data requires this preprocessing, so it is interesting to see if the created models are affected by the standardisation of the data, improving or worsening the predictions of new samples.

For the same reasons as in the previous tests, the data used for Test 6.1 and 6.2 have selected to be able to compare the results. The standardisation of the samples follows Equation 3.1. Once the data is standardised it is separated into training, validation and test data as in the other tests, and then applied to the methods.

The results for this test can be seen in Section 7.6.

CHAPTER 7

Results

In this chapter, the results for the tests described in Chapter 6 will be presented. For each of the tests, both methods of dimensionality reduction have been carried out prior to the training of the SVM classifier.

The results of each method will be separated into subsections within each test. First, showing the amount of variance collected in each of the eigenvectors, followed by a plot of the results from the classifier where the hyperplane is created along with its margins and the accuracy of the unbiased test data.

7.1 Results Test 1: Classification of an easy dataset

In this section, the results from Test 1 described in Section 6.1 will be presented. This test is meant to show how the classifier performs when the faulty data is easily separable from the non-faulty data. This can be seen in Figure 6.1 where the samples from 1 to 3000 have different values compared to the samples from 3001 to 6000.

7.1.1 PCA SVM Test 1

The variance collected in each of the eigenvectors for PCA in this test can be seen on the scree plot in Figure 7.1. It is shown that the first two Principal Components collectively have more than 99% of the variance from the data.

Once the PCA is performed and the data is reduced into 2 dimensions, the SVM is

trained and the test data is classified. The results for this test are shown in Figure 7.2 where the red dots refer to the Non-Faulty training data and the blue squares are the Faulty training data. While the Non-Faulty test data are shown as green dots and the Faulty test data are the yellow squares. The Support Vectors are shown with black rings and the hyperplane is a black line with the margins being the dashed lines.

Since all tested C values had 100% accuracy for the validation data, $C = 100$ has been selected, which has a good amount of support vectors without going into the margins. The testing data also achieved an accuracy of 100% which was expected since the data was easily separable

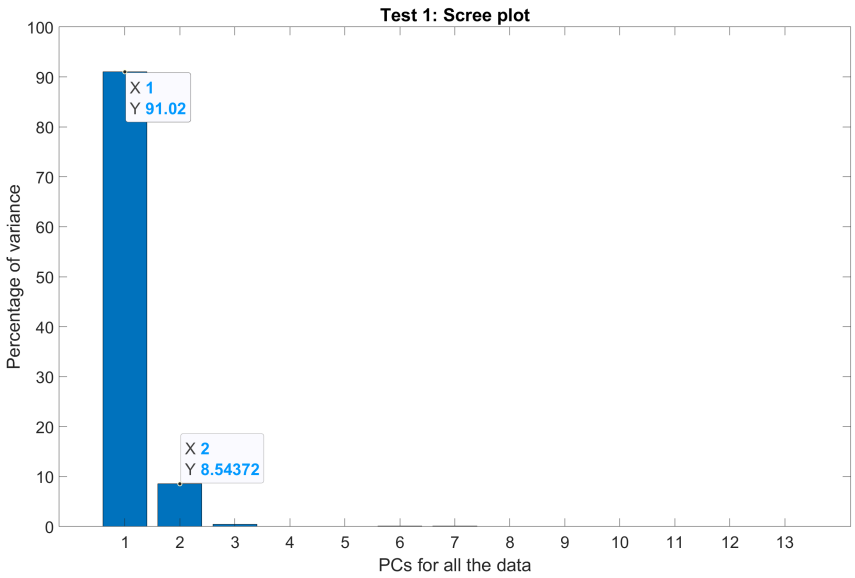


Figure 7.1: Scree plot for PCA method in Test 1

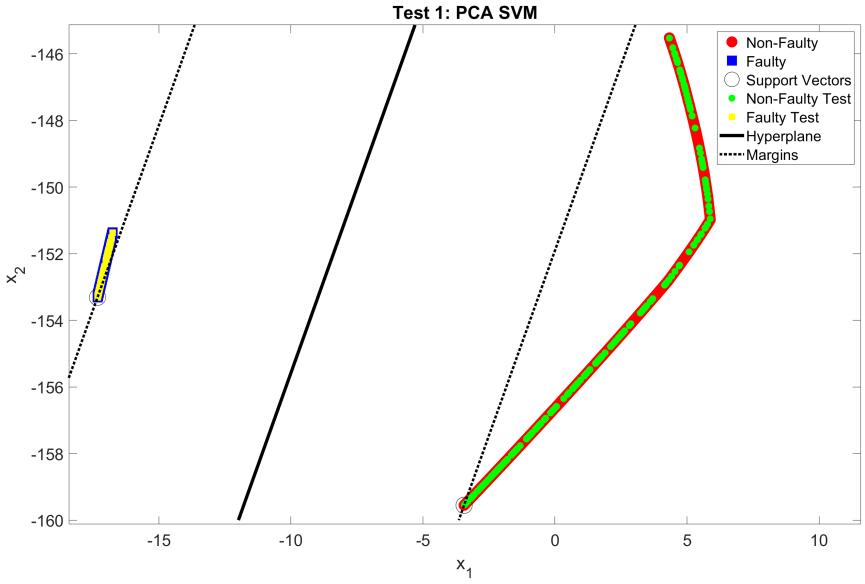


Figure 7.2: Hyperplane and margins for PCA SVM Test 1

7.1.2 LDA SVM Test 1

The scree plot for the LDA method can be seen in Figure 7.3. It is worth mentioning that it would only be necessary to use the first eigenvector, since it collects 100% of the variance making the rest of them negligible. It has however been decided to use two eigenvectors for the sake of consistency throughout the figures and to facilitate the visibility of the samples.

For this test, the SVM created can be seen in Figure 7.4. The fact that only one eigenvector is sufficient for separating the data means that the second direction provides little to no information.

Since all the samples lie on the margins, the selection of the C parameter does not have any effect on the SVM model. Especially when it comes to finding the support vectors, since practically all the samples are at the same distance from the hyperplane. The regularisation parameter is therefore chosen to be $C = 100$, since not many support vectors are necessary for defining the margins. The results from both the validation and testing data show 100% accuracy.

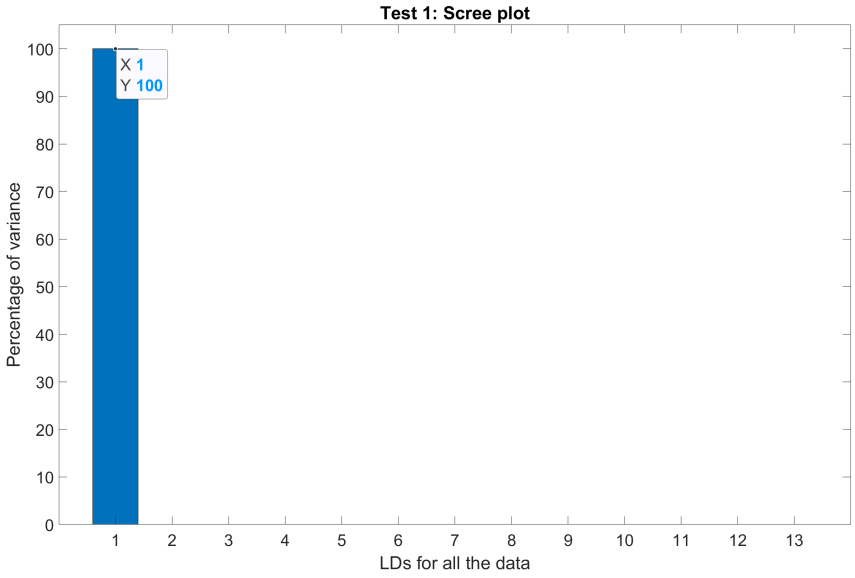


Figure 7.3: Scree plot for LDA method in Test 1

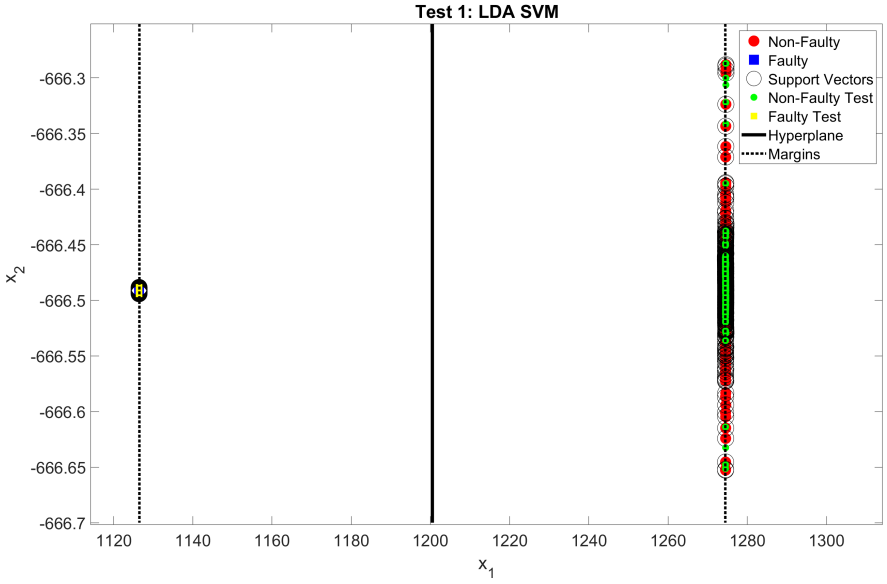


Figure 7.4: Hyperplane and margins for LDA SVM Test 1

7.2 Results Test 2: Classification of a difficult dataset

The results for Test 2 in Section 6.2 will be described here. This test aims to show how well the methods handle data that is not easily separable.

7.2.1 PCA SVM Test 2

The scree plot in Figure 7.5 shows that with the first 2 PC's more than 98% of the variance is collected. However, since PCA does not consider the labels of each sample, the projection of the data on these two main directions causes both classes to overlap. This can be seen in Figure 7.6 where the data from both classes intermingle. Which is also reflected in the accuracy of the test data, as it is 50.50%. Specifically for the model represented in Figure 7.6, the optimal regularisation parameter C is 0.1 and the validation data has an accuracy of 51.25%. These percentages are approximations as they depend on the randomisation when splitting the classes.

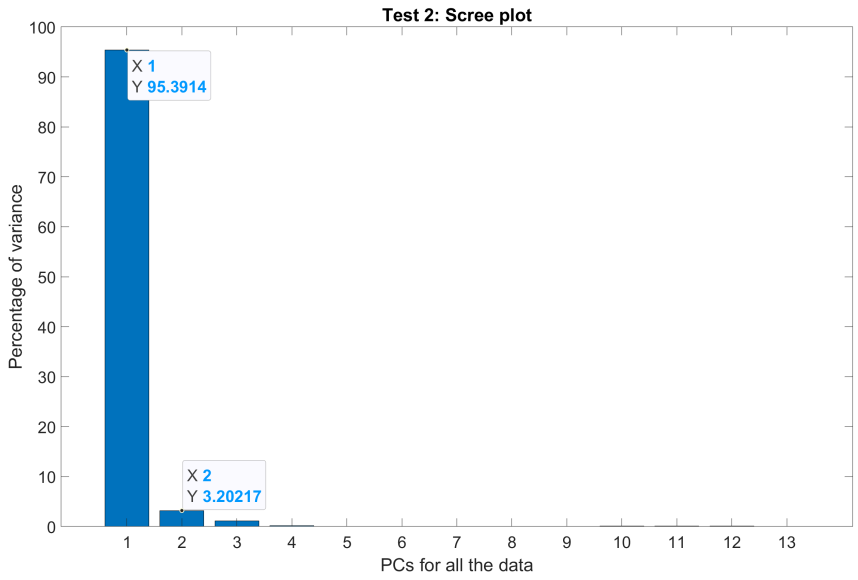


Figure 7.5: Scree plot for PCA method in Test 2

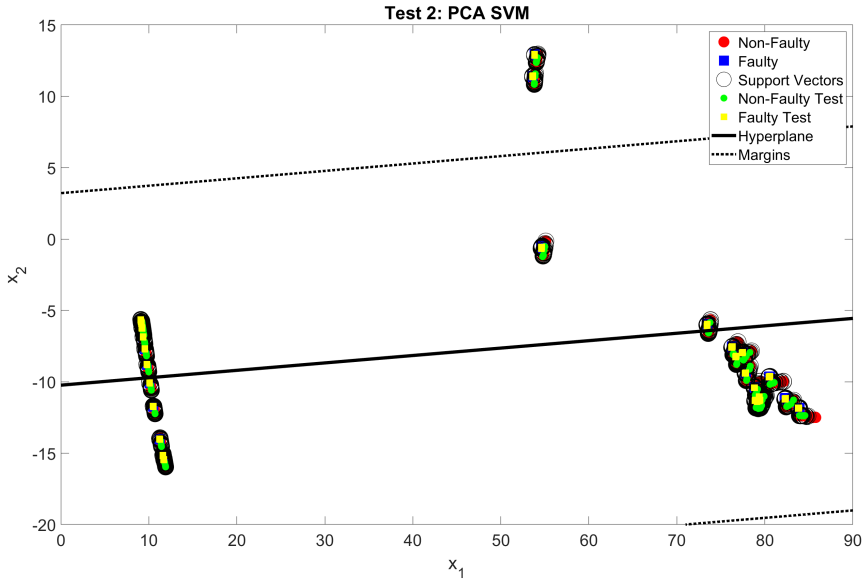


Figure 7.6: Hyperplane and margins for PCA SVM Test 2

7.2.2 LDA SVM Test 2

As in Test 1 for the LDA method, the scree plot for this test shows 100% variance for the first eigenvector which can be seen in Figure 7.7. However, as expected when projecting the classes, they are not as easily separable. In Figure 7.8 it can be seen that a few samples cross the hyperplane when creating the SVM model. When training the model, the optimal regularisation parameter C was 100, with a validation data accuracy of 97.12%. The test data performed similarly achieving an accuracy of 96.75%

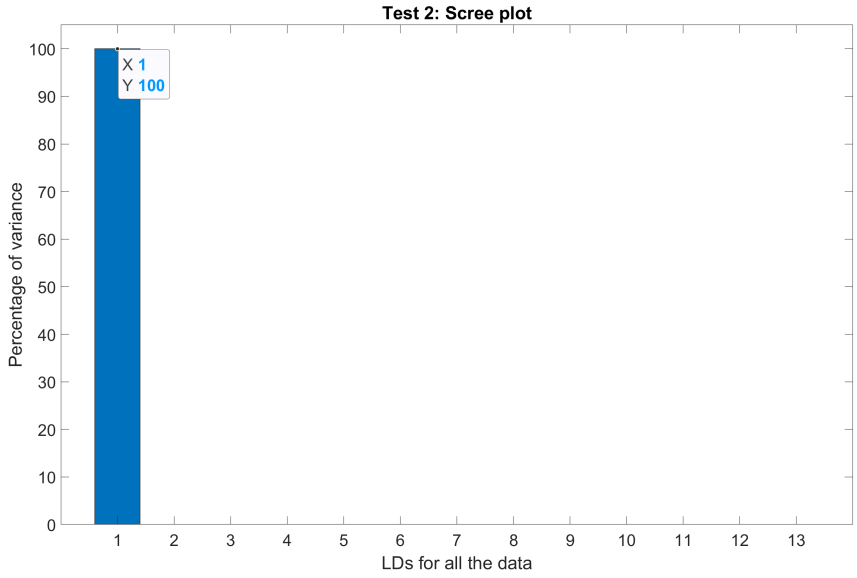


Figure 7.7: Scree plot for LDA method in Test 2

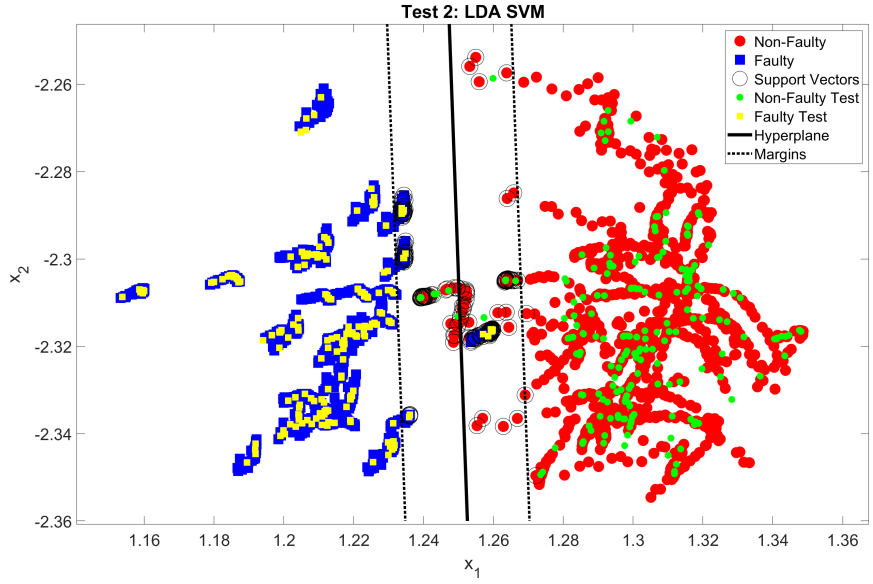


Figure 7.8: Hyperplane and margins for LDA SVM Test 2

7.3 Results Test 3: Effect of noise on the data

The results of adding noise to the data are going to be explained in this section. As mentioned in Section 6.3 the noise has been applied to the training, validation and test data, by adding values to the samples with $\mathcal{N}(0, 2)$.

7.3.1 PCA SVM Test 3

Adding noise to the training data from Test 6.1 showed that at least 4 PC‘s are needed to collect a variance greater than 90% in the scree plot of Figure 7.9. However it was still possible to achieve 100% accuracy in the test and validation data using the first two Principal Components, as seen in Figure 7.10 with $C = 100$. The limits of this test has been tested by increasing the variance of the noise, which resulted in the model still being able to classify the data with a variance far beyond what a real system would ever encounter.

In the second test, where the noise is added to the testing data the first two PC´s gathered a variance of 99%. From this test a prediction and validation accuracy of 100% is achieved with $C = 100$ as shown in Figure 7.11. The prediction accuracy however decreases once the variance of the noise is greater than 4.

For the third test the same data is used but the noise is added to both the training and testing data leading to a similar scree plot seen in Figure 7.9. The noise in both the training and testing data does not have a sufficient enough negative effect on the predictions as can be seen in Figure 7.12. This is due to the clouds of the testing data remaining on either side of the hyperplane. The model is more affected when the noise is added to the test data compared to the training data.

The same results were achieved when adding noise to the training, validation and testing data. This was however expected as the noise in the validation data only affects the training phase of the SVM, which in some cases reduces the accuracy of the validation data but not the overall performance of the model. It has been tested that even increasing the variance of the validation noise beyond a realistic level, the accuracy of the testing data is not affected.

Testing the effect of noisy data on a simulation considered as difficult for separating the classes is deemed unreasonable as the results shown in Test 7.2.1 proved that the methodology is not reliable for these cases.

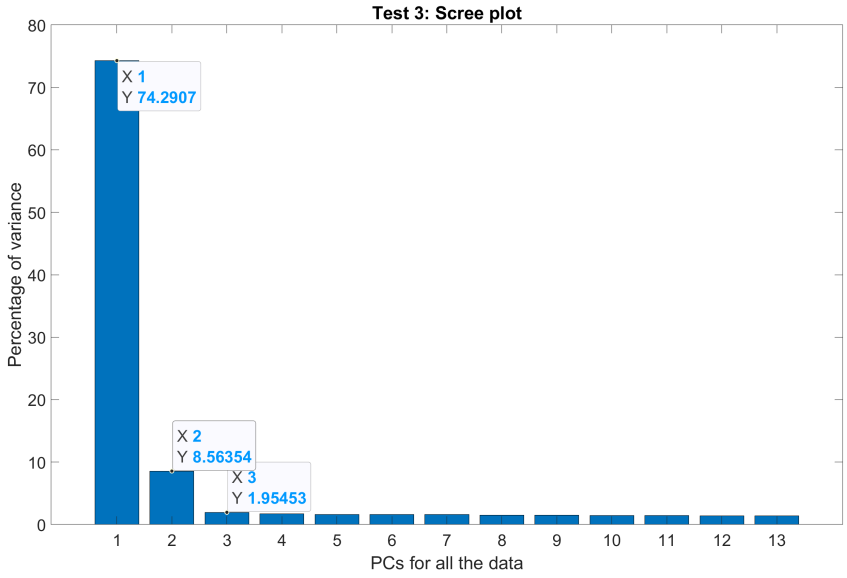


Figure 7.9: Scree plot for PCA method in Test 3 with noise in the training data

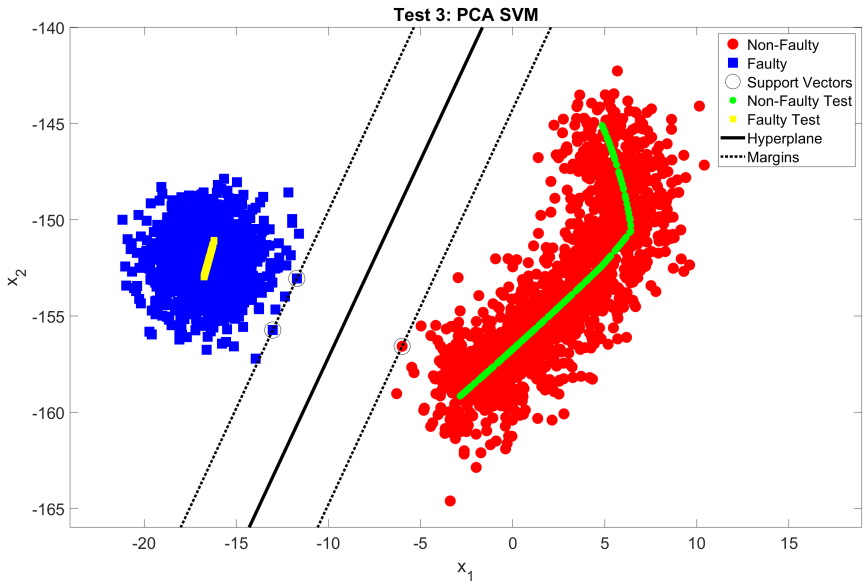


Figure 7.10: Hyperplane and margins for PCA SVM Test 3 with noise in training data

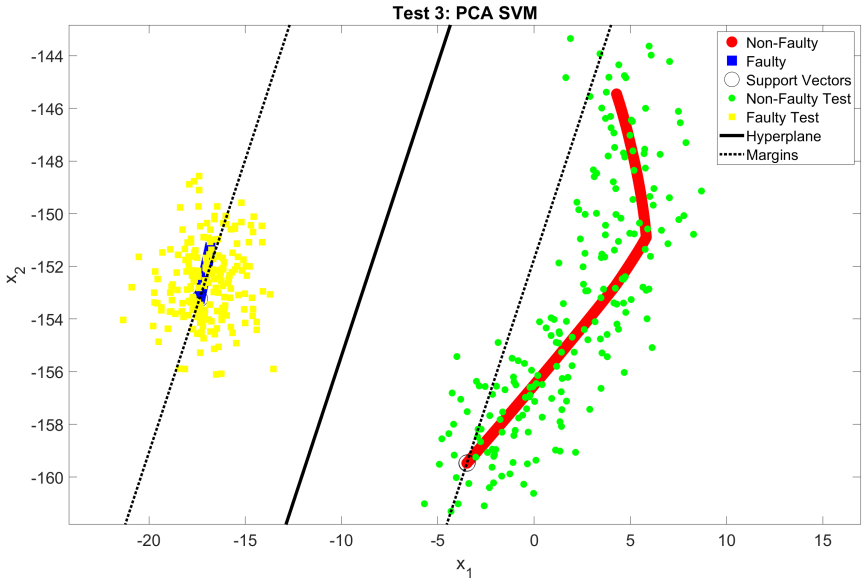


Figure 7.11: Hyperplane and margins for PCA SVM Test 3 with noise in testing data

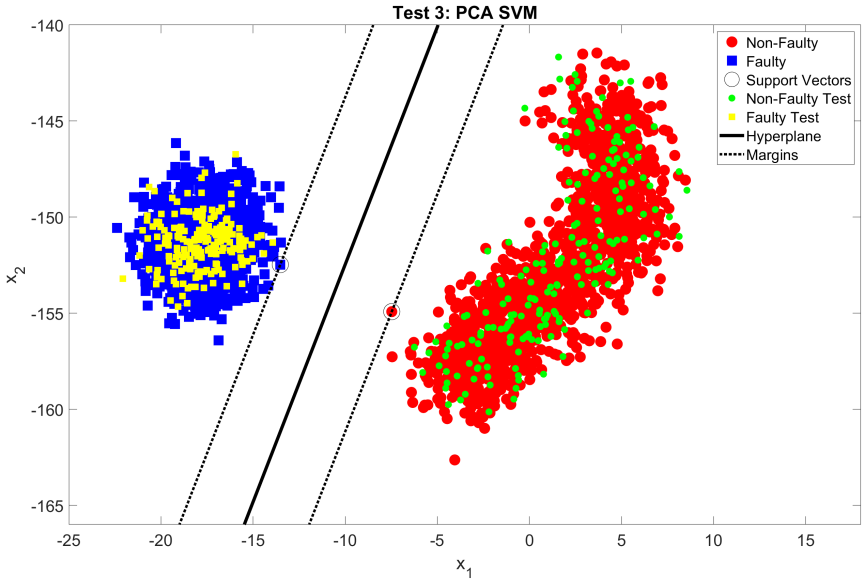


Figure 7.12: Hyperplane and margins for PCA SVM Test 3 with noise in training and testing data

7.3.2 LDA SVM Test 3

The scree plots for all the tests done for LDA SVM in Test 3 showed 100% variance in the first eigenvector.

In Figure 7.13 noise is applied to the training data for simulation 30 achieving an accuracy of 100% for both the validation and testing data. However, when the noise is applied to the testing data for the same simulation, as seen in Figure 7.14, the testing accuracy drops to 50% while the validation accuracy remains the same. Illustrating how much the model is affected by noise from the testing data. However, when the SVM model is both trained and tested with noisy data as seen in Figure 7.15 it can once again achieve an accuracy of 100% in both the validation and testing data. Showing that the model becomes more robust against noise when it is trained on noisy data.

When applying noise to the data from Test 6.2, the model is also affected by noise in the training data. In Figure 7.16 the model is shown with noise applied to the training data, achieving an accuracy of 85.32% in validation data and 86.25% in the testing data for $C = 100$. The results worsened when applying noise to the test data. As can be seen in Figure 7.17 the clouds of test data intermingle and cross the hyperplane, decreasing the accuracy of the testing data to 69.05%.

Finally, noise was also added to the validation data to see whether it would have any effect on the accuracy. The accuracy for the test data remained almost the same at 69.79%, while the validation accuracy decreased significantly to 68.25%. This was expected as the noise in the validation data only affects the training phase, which does not have a big impact on the testing data. The small differences in the percentages are due to the randomly generated noise and are therefore negligible.

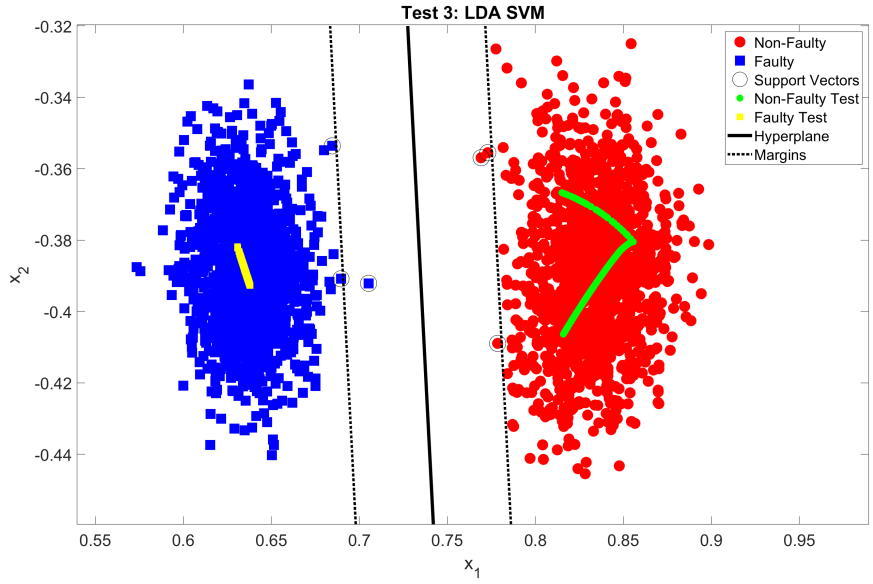


Figure 7.13: Hyperplane and margins for LDA SVM Test 3 with noise in the training data from simulation 30

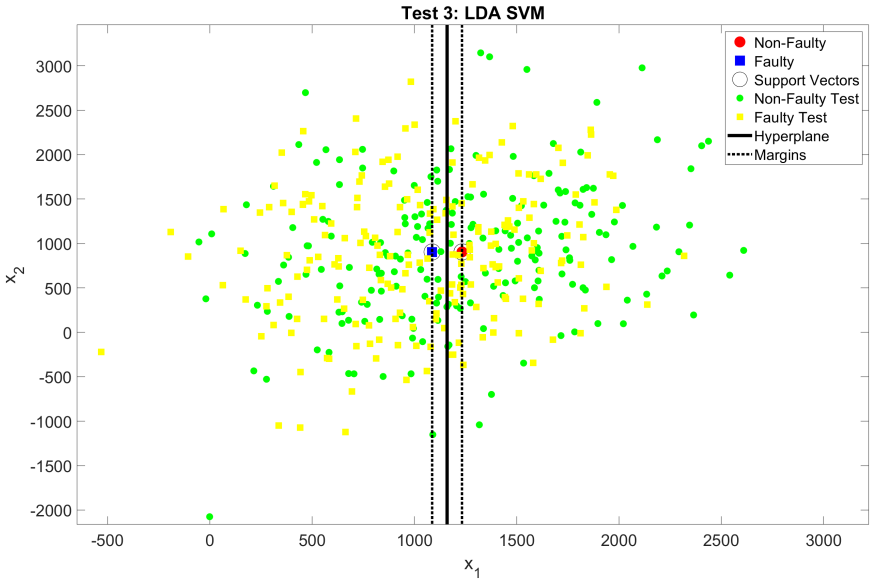


Figure 7.14: Hyperplane and margins for LDA SVM Test 3 with noise in the testing data from simulation 30

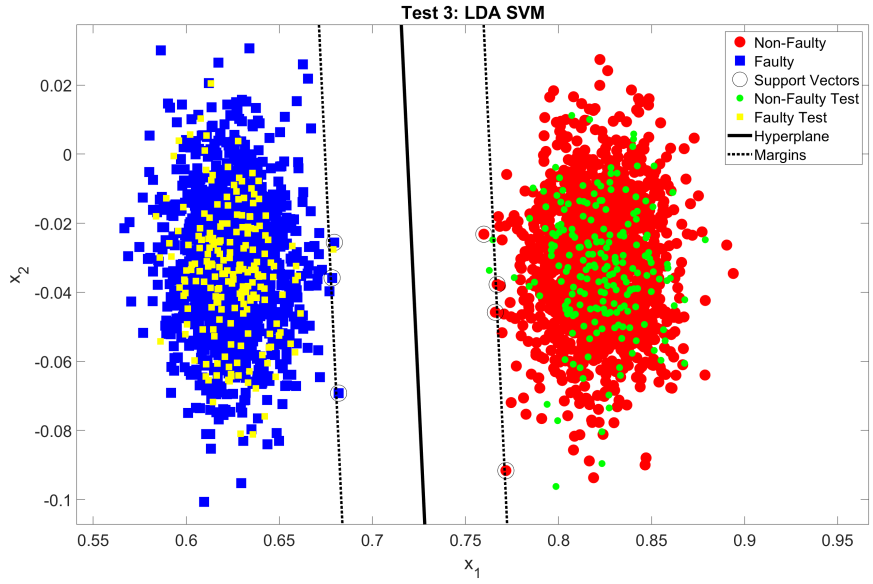


Figure 7.15: Hyperplane and margins for LDA SVM Test 3 with noise in the training and testing data from simulation 30

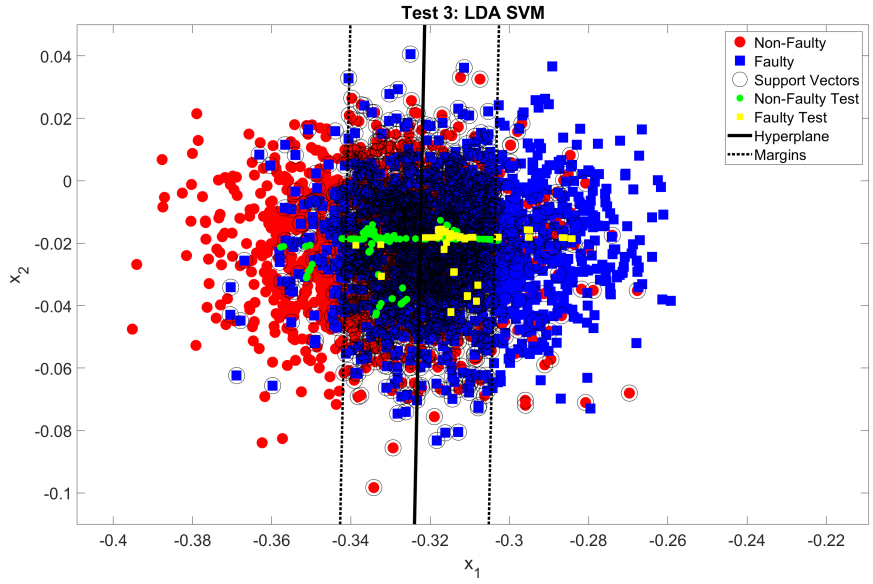


Figure 7.16: Hyperplane and margins for LDA SVM Test 3 with noise in the training data from simulation 21

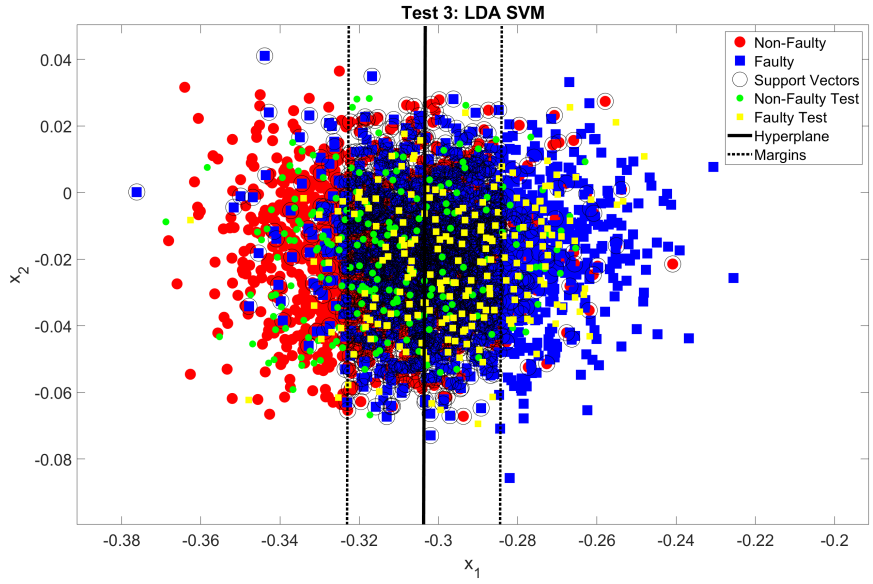


Figure 7.17: Hyperplane and margins for LDA SVM Test 3 with noise in the training and testing data from simulation 21

7.4 Results Test 4: Using a different simulation as testing data

The results of predicting test data from a different simulation will be described in this section. This test is only carried out for the simulations which are easily separable as the other simulations already have low accuracy.

Since the validation data is extracted from the same simulation as the training data, the accuracies of the validation data for all the different tests carried out in this section are 100%.

7.4.1 PCA SVM Test 4

The scree plots for all the tests using PCA achieved approximately 99% of the variance for the first two Principal Components.

The model using the testing data extracted from simulation 22 can be seen in Figure 7.18. It presents an accuracy of 82.75% for this particular testing data.

When using the model with the testing data from simulation 36, as seen in Figure 7.19, the accuracy increases to 100%.

This is however not the case for all simulations as it was confirmed using testing data from simulation 38 that the model was unable to classify the data reaching a testing accuracy of 57.50%. This shows that the classifier can only classify testing data from a different simulation if it is similar enough to the training and validation data.

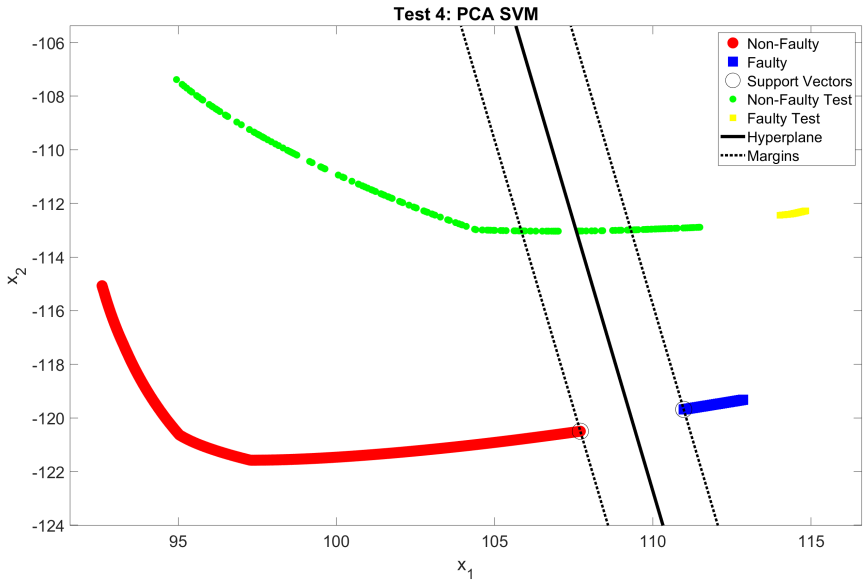


Figure 7.18: Hyperplane and margins for PCA SVM Test 4 using test data from simulation 22

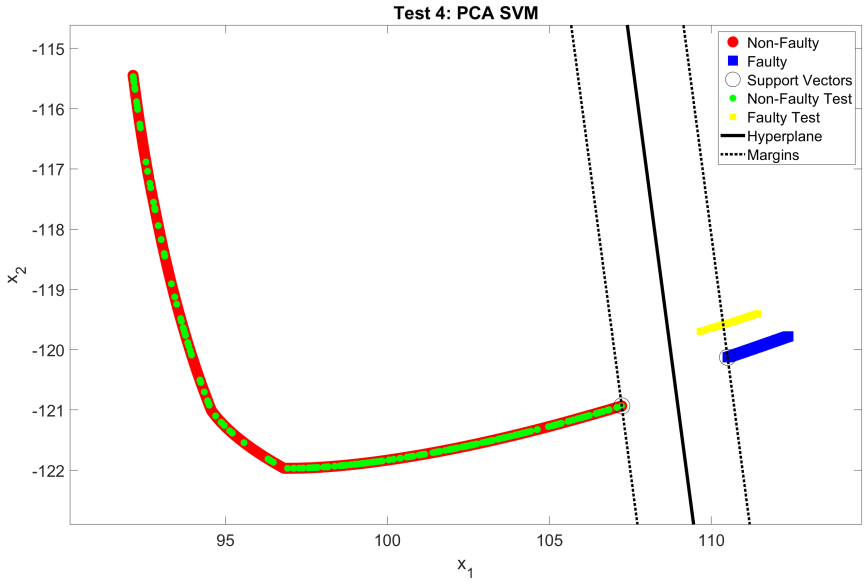


Figure 7.19: Hyperplane and margins for PCA SVM Test 4 using test data from simulation 36

7.4.2 LDA SVM Test 4

All the tests using the LDA achieved an accuracy of 100% in the first eigenvector of the scree plot.

As seen in Figures 7.20 and 7.21 the test data is found in its correct side of the hyperplane, thus the predictions have an accuracy of 100% in both models. This model has also been confirmed using simulation 38 reducing the accuracy to 50%. Illustrating that not all different simulations can be classified using this model.

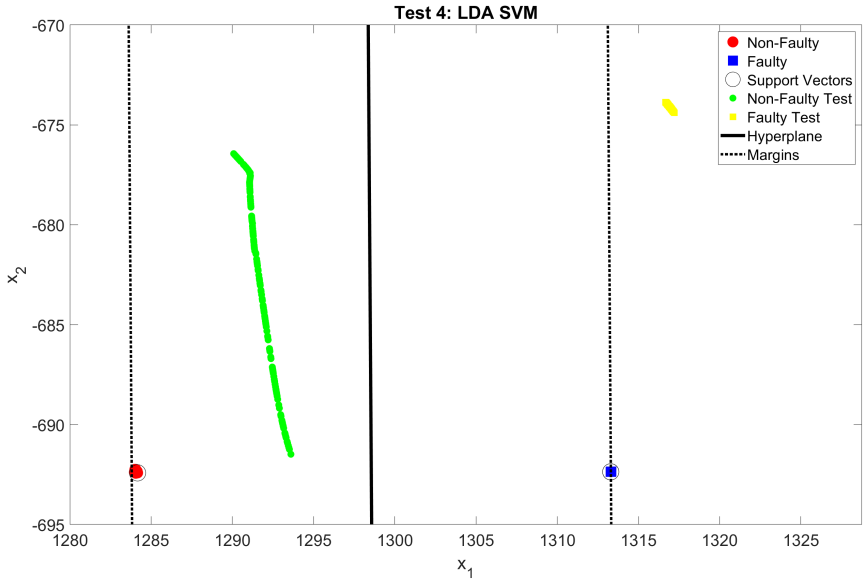


Figure 7.20: Hyperplane and margins for LDA SVM Test 4 using test data from simulation 22

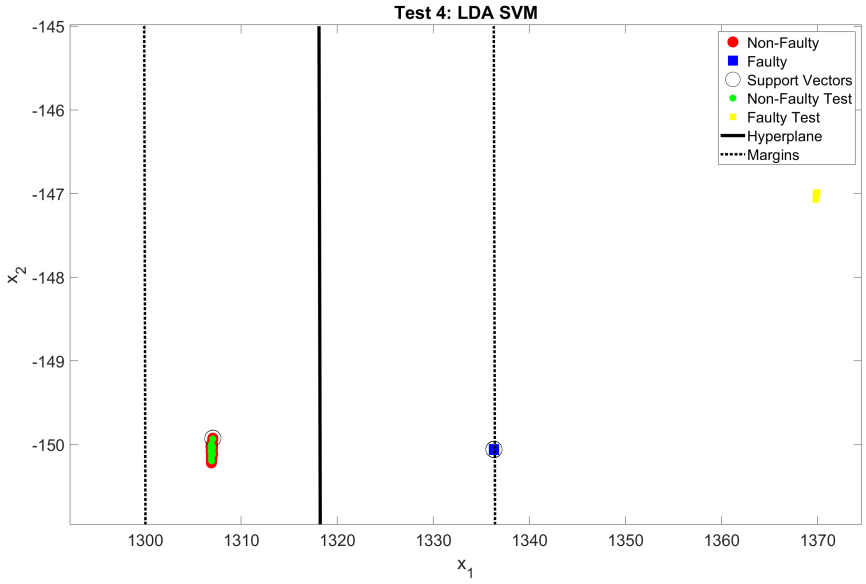


Figure 7.21: Hyperplane and margins for LDA SVM Test 4 using test data from simulation 36

7.5 Results Test 5: Effect of the dynamics in the signals

In this section, the results using the classifier with the dynamic part of the signals are going to be described. The dynamics of the signals worsen the consistency in the signals making the classification of the data more difficult.

The scree plots are not going to be shown since they are very similar to the previously shown figures, however the variance collected in the main feature directions are going to be mentioned.

7.5.1 PCA SVM Test 5

The variance collected from the data in Test 6.1 with the dynamics is 88.85% and 9.16% for the first two PC's. While the variance from the dynamical data in Test 6.2 is 94.69% and 3.35%.

The results of the classifier with the dynamics are shown in Figure 7.22. Comparing it to the results without them in Figure 7.2 shows that the dynamics make it harder for the model to classify data. This can be seen as the validation and test accuracy decreases from 100% to 95.87% and 100% to 97.50% respectively for $C = 10$.

Similar results are obtained when using data from a simulation with overlapping samples in both classes. In this case, the scree plot has a variance of 94.69% and 3.35% in the first two PC's. From the SVM model in Figure 7.18, a validation accuracy of 48.75% and a test accuracy of 49.75% are achieved with $C = 100$ making the model unreliable.

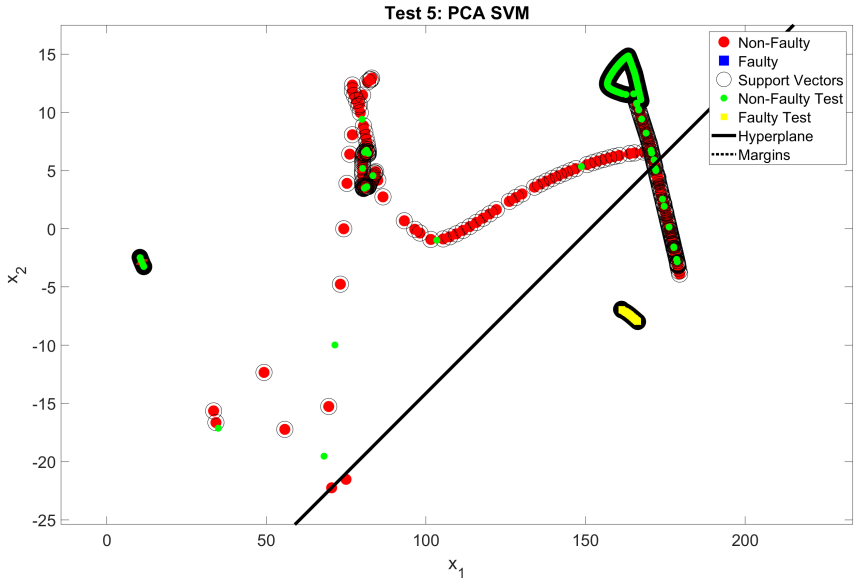


Figure 7.22: Hyperplane and margins for PCA SVM Test 5 including dynamics from simulation 30

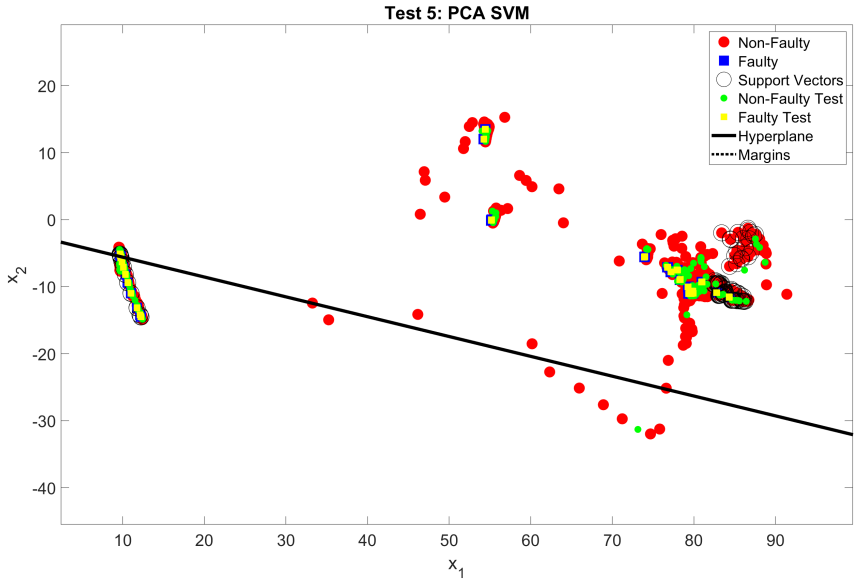


Figure 7.23: Hyperplane and margins for PCA SVM Test 5 including dynamics from simulation 21

7.5.2 LDA SVM Test 5

Although the model is somewhat affected by the dynamics, the LDA SVM method continues to maintain high levels of success in both cases. The scree plots for both data maintain 100% variance in the first eigenvector.

The models including the dynamics for the two different simulations are shown in Figures 7.24 and 7.25. In the first model the classes are still easily separable, achieving 100% accuracy for both the validation and test data for $C = 100$. Whereas in the second model the accuracy decreases for both the validation data and test data going from 97.12% to 96.88% and 96.75% to 96.25% respectively in reference to the model without the dynamics.

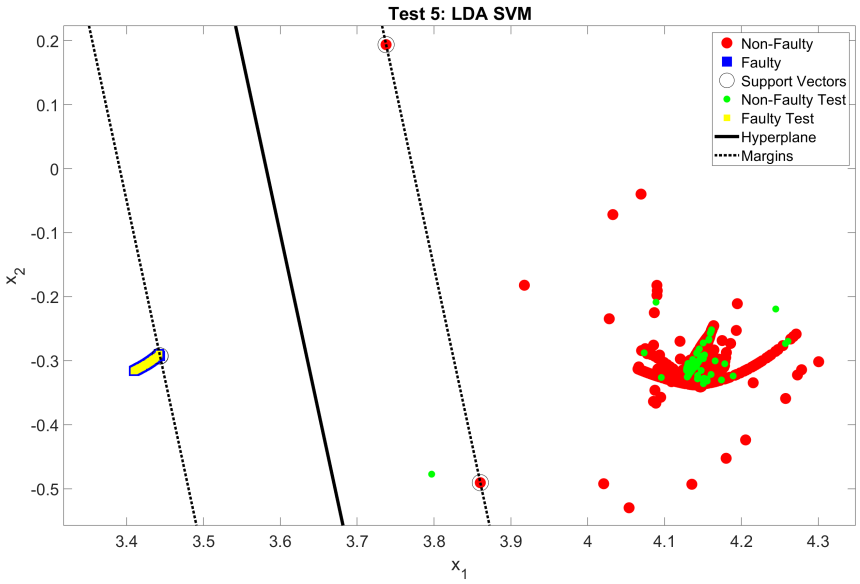


Figure 7.24: Hyperplane and margins for LDA SVM Test 5 including dynamics from simulation 30

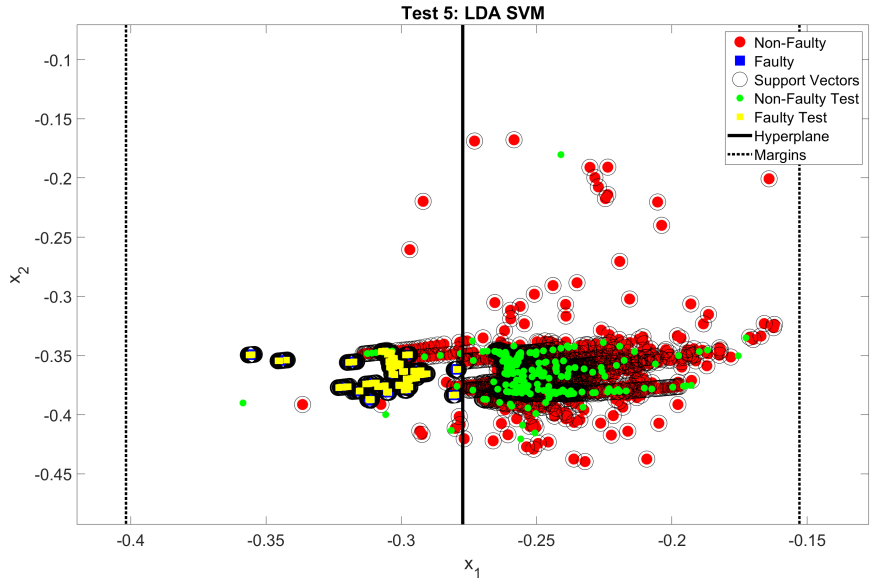


Figure 7.25: Hyperplane and margins for LDA SVM Test 5 including dynamics from simulation 21

7.6 Results Test 6: Effect of standardising the data

The results of this test differ a little from those previously obtained in Test 7.1 and 7.2.

7.6.1 PCA SVM Test 6

The PCA method with the data from Test 6.1 looks very similar to the first test itself. The scree plot reaches 99% variance in the first two PC's and the SVM model has 100% accuracy in both, validation and testing data. The main difference in this case is that the data remains centred.

There are however some differences in the PCA method with the data from Test 6.2. The scree plot of the standardised data can be seen in Figure 7.26 showing that the variance is spread along more PC's, requiring a total of 4 PC's to achieve more than 90%. For this reason, the SVM model can not be shown in a figure, but the hyperplane can still be computed and the test data can be predicted. The results from this test gets a validation accuracy of 48.63% and a test accuracy of 51.50%

similarly to the results from Test 7.2.

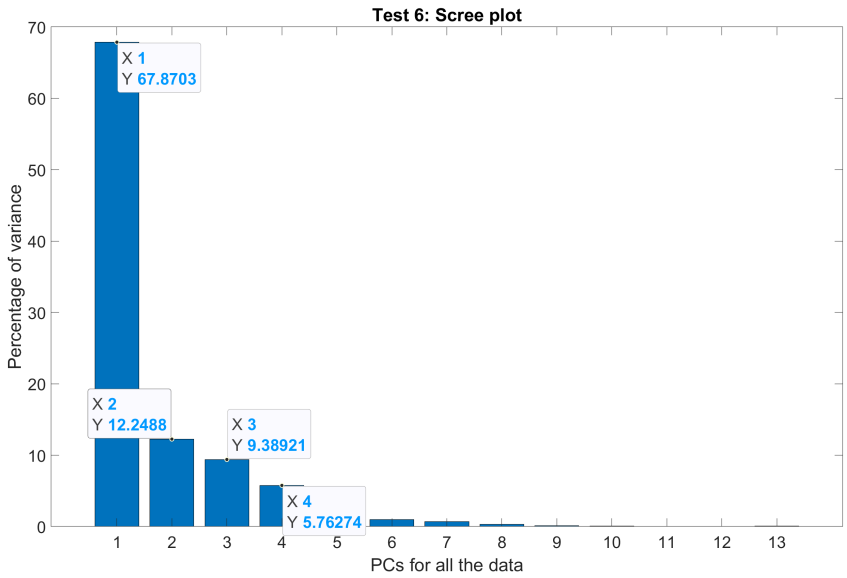


Figure 7.26: Scree plot for PCA method in Test 6

7.6.2 LDA SVM Test 6

For the LDA method, the only difference that can be underlined is that the data is centred after the dimensionality reduction. Both scree plot and accuracy remain the same as in Tests 7.1 and 7.2 with the only exception being the testing accuracy of 97.25% for simulation 21.

7.7 Overall results

In this section, an overall overview of the results from this chapter have been collected in Table 7.1 for the different tests carried out in order to facilitate the comparison of the test accuracies for both methods. In this table, the ($^\diamond$) after the percentage refers to the noise applied to the testing data while the (*) is when noise is applied to both, training and testing data and finally, the (†), refers to the test with noise in the whole dataset.

Table 7.1: List of all the test accuracies for both methods

Test No.	Dataset	PCA	LDA
1	Simulation: 30	100%	100%
2	Simulation: 21	50.50%	96.75%
3	Simulation: 30	100%	100%
		100%$^{\diamond}$	55%$^{\diamond}$
		100%*	100%*
		100%†	100%†
	Simulation: 21	[-]	85.32%
		[-]	69.05%*
		[-]	69.79%†
4	Simulation: 24 & 22	82.75%	100%
	Simulation: 24 & 36	100%	100%
	Simulation: 24 & 38	57.50%	50%
5	Simulation: 30	97.50%	100%
	Simulation: 21	49.75%	96.25%
6	Simulation: 30	100%	100%
	Simulation: 21	51.25%	97.25%

CHAPTER 8

Conclusion

The objective of this project was to develop and implement an automatic fault detection model for a supermarket refrigeration system without the necessity of manually monitoring all the different signals from it. To do this, in Section 4.3 a classifier has been developed using the Support Vector Machine method, which finds a hyperplane that divides the space and separates the samples from the signals when the system is in normal operation and in a failure state. To facilitate the work of the classifier, and to be able to show the results in a 2D graph, a reduction of the dimensions has been applied to the original data using two methods: an unsupervised method (PCA) in Section 4.1 and a supervised method (LDA) in Section 4.2.

The dimensionality reduction methods and the classifier have been successfully implemented and several tests have been carried out in Chapter 6 to verify and validate them under different conditions. In these tests, the methods developed using data obtained from the simulations have been tested. Faulty samples have been detected against non-faulty samples having both similar and different values. The limits of the developed models have been verified: by adding noise to the training, validation and testing data in Section 6.3; using test data from different simulations in Section 6.4; including the dynamic part of the signals in Section 6.5; and finally, in Section 6.6 the effects of standardising the data prior to reducing its dimensions have been verified.

All the results from Chapter 7, are collected in Table 7.1 to show that, although both methods are appropriate for reducing dimensions, in general, using LDA presents a greater success when it comes to predicting faulty data. This was expected from the beginning since PCA does not distinguish between the classes, and treats all samples as a single dataset, whereas LDA focuses on separating the classes when reducing the dimensions, which sets up a perfect distribution of the samples for the SVM classifier.

From the comparison of Test 6.1 and 6.2 can it be concluded that the models trained using simulations with high values of $HeatLoad$ ($O(10^4)$) perform better in comparison to low values ($O(10^3)$). This is to be expected since the signals are more constant with high values and oscillate more with low values.

The tests concerning the addition of noise show that when it is applied to the training and validation data, the model is barely affected as the dimensionality reduction methods seem to reduce the impact of it, even with high values of noise variance. However, when it is applied to the test data, the prediction accuracy drops when the variance of the noise is greater than 3. The prediction performance improves when the model is trained with noisy data in comparison with the original data from the simulation. It can be concluded from the fourth test that training the classifier model with one simulation fault does not assure that the prediction of different faults will be detected since it is highly dependant on how the signals are in comparison with the training data.

The dynamics of the signals affect both methods, as the test accuracies decrease compared to the tests without it. The PCA method is however more influenced by the dynamics as the drop in the accuracy is higher.

The last test showed that standardising the data prior to the classification is not required as it does not have an impact on the predictions.

It can be concluded from all the tests that the LDA SVM method presented in Section 4.4.2 is more robust against changes in the training data, generally allowing greater noise and being less affected by the dynamic part of the signals.

The classification can be run in real-time with new data as it only requires two steps which are computationally effortless. The first step is preprocessing the data using the transformation matrix extracted from either PCA or LDA and then calculating the sign of Equation 3.19 to predict whether it is a faulty sample or not. Additionally, the model is very flexible adapting to the current system and being able to be trained with the available signals making it applicable for various SRSs.

CHAPTER 9

Discussion and future work

It is evident from the conclusion in Chapter 8 that the LDA method outperforms PCA for classification of the classes. However this conclusion is purely based on the results and tests carried out which do not completely reflect the general performance of the methods for all faulty scenarios. Before concluding that one method is better than the other, one could say that a more comprehensive analysis should be done. This could involve testing for all the different faulty signals to verify that it performs, on average, better than the other. However this task would be very time-consuming and it was instead preferred to test the methods under different conditions.

The noise in the third test has been chosen to be $\mathcal{N}(0, 2)$, which was deemed appropriate for the system as this magnitude is bigger than what will be encountered in the field. It was expected from the PCA method that it would barely be affected by the noise as it extracts the main feature directions with the most variance. A similar outcome appeared for the LDA method as it compresses the noise inside of each class. Noise in data from a real system can not be avoided and therefore training a model with noisy data will make it more robust against predicting actual data.

Trying to classify data from a different simulation than the model is trained on is very dependent on, the similarities between the data. From the tests completed in Section 7.4 a concrete conclusion can not be formed as it is based on a comparison between three different tests which do not fully show the correlation between the T_{set}, $HeatLoad$ and different fault signals. To achieve a more comprehensive conclusion several comparisons between the different simulations should be done in order to analyse the impact of each parameter in the signals.

It has not been tested whether the models suffer from overfitting, which occurs when they are specifically trained for a certain dataset making them biased towards new

data and unable to classify it correctly. In all the tests, the models are trained with the same number of samples to maintain consistency and to be able to compare the results from one test to another. New tests would have to be carried out by varying the number of samples used for training the models and observing its effect on the prediction of new samples. A way of reducing the effects of overfitting is by applying noise to the training samples, which is shown in Test 3 with a less biased model.

9.1 Future work

This section proposes improvements to the implemented methods which could lead to better results. However these implementations were not feasible within the time frame of this thesis and are therefore listed as future work. This includes SVM for multiple classes, kernels, cross validation and combination of multiple simulations for training.

9.1.1 SVM for multiple classes

The current implementation of the SVM model is only capable of classifying a two class problem. Whereas a different strategy could be implemented to classify $K > 2$ classes. In this case the idea is to be able to expand the amount of faults that the classifier can predict. Instead of only classifying whether the data is faulty or not, it could potentially also predict which signal is faulty. In Section 3.4.4 three different approaches for this implementation were proposed. The amount of iterations for comparing all the faults in the first two approaches increases quadratically to the number of classes, it will slow down the process to the point that it could not be predicted in real-time. The last approach, DAGSVM reduces considerably the amount of iterations needed to classify the sample among all the classes, which could be implemented in a real system.

9.1.2 Kernels

The results from the classifier could potentially be improved by applying the kernels mentioned in Section 3.4.3.1. In general, when using LDA as a dimensionality reduction method, the data is transformed in such a way that it is not necessary to implement a kernel. However, when noise is applied to the data, the prediction results could be improved by implementing a Gaussian kernel, which gives the model greater flexibility by not limiting it linearly.

9.1.3 Cross validation

It could be used to make a more robust model as it uses the whole dataset for training and validation. This could be beneficial in scenarios where the dataset is smaller. One of the ways is to use the K-fold cross validation. Here, the entire dataset is divided into K packets with the same number of samples. K iterations are performed in which one of the packages is used as validation data, and the rest as training data. In each iteration, the package used as validation data is changed, and finally the optimal model is obtained by making an average of the results.

9.1.4 Combination of multiple simulations for training

The idea behind this strategy is to use multiple simulations in the training phase of the SVM and then create a more general model which can be used for different faults. To carry out this model, random samples from different simulations would be extracted forming the training and validation matrices which will be used in the methods PCA/LDA and SVM. The idea is to mix simulations that are different enough to create a more flexible model, but that are not too different since those samples could be considered as outliers.

Bibliography

[1] Z. Soltani et al. "Fault Detection of Supermarket Refrigeration Systems Using Convolutional Neural Network". In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. 2020, pp. 231–238. DOI: 10.1109/IECON43393.2020.9254485.

[2] Z. Soltani et al. "PCA-SVM Data Classification of Supermarket Refrigeration Systems-Robustness Analysis". In: (2021).

[3] JUSTIN KAUWALE. *Pressure Enthalpy Diagram*. Uploaded: 2020. URL: https://www.engproguides.com/pressure-enthalpy-diagram.html.

[4] Dave Demma. *The Pressure - Enthalpy Chart*. Uploaded: 01-2005. URL: https://sporlanonline.com/literature/education/5-200.pdf.

[5] Moe. *Heat Pump Refrigeration Cycle (II)*. Uploaded: 2009. URL: http://www.heatpump-reviews.com/refrigeration-cycle.html.

[6] Jake Krzywinski et al. *Principal component analysis*. Uploaded: 2017. URL: https://www.nature.com/articles/nmeth.4346.

[7] Zakaria Jaadi. *A STEP-BY-STEP EXPLANATION OF PRINCIPAL COMPONENT ANALYSIS*. Updated: 05-12-2020. URL: https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[8] Ian T. Jolliffe and Jorge Cadima. *Principal component analysis: a review and recent developments*. Updated: 13-04-2016. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202.

[9] S. Ji and J. Ye. "Generalized Linear Discriminant Analysis: A Unified Framework and Efficient Model Selection". In: *IEEE Transactions on Neural Networks* 19.10 (2008), pp. 1768–1782. DOI: 10.1109/TNN.2008.2002078.

[10] Zheng-Hua Tan. *Lecture 6: Linear Classification Methods*. Updated: 07-10-2020. URL: https://www.moodle.aau.dk/pluginfile.php/2132917/mod_resource/content/1/ML_6_Linear_discrimination.pdf.

[11] Alaa Tharwat et al. "Linear discriminant analysis: A detailed tutorial". In: *Ai Communications* 30 (May 2017), pp. 169–190. DOI: 10.3233/AIC-170729.

[12] Alaa Tharwat. *Linear Discriminant Analysis: An Overview*. Updated: 30-12-2017. URL: https://www.researchgate.net/publication/289528785_Linear_Discriminant_Analysis.

List of Figures

List of Tables

APPENDIX A

Matlab functions

A.1 PCA function

```
function [U, DsPCA, VsPCA] = PCA (rawdata, nPC)
D = size(rawdata,1);
N = size(rawdata,2);
mu = mean(rawdata,2);

% PCA Step 1: Covariance Matrix S
S = zeros;
for i = 1 : N
    S = S + (rawdata(:,i) - mu)*(rawdata(:,i) - mu)';  %eq: 3.3
end
S = S/N;

% PCA Step 2: Eigenvectors and eigenvalues of S
[evecPCA,evalPCA]=eig(S);   %eq: 3.4

% PCA Step 3: Sorting and finding the highest eigenvalues
[dPCA,indPCA] = sort(diag(evalPCA),'descend');
DsPCA = evalPCA(indPCA,indPCA);
VsPCA = evecPCA(:,indPCA);

% PCA Step 4: Transformation matrix U
u= [VsPCA(:,1) VsPCA(:,2) VsPCA(:,3) VsPCA(:,4) VsPCA(:,5) VsPCA(:,6)];
U = u(:,1:nPC)';
```

Source A.1: The PCA function used for dimensionality reduction

A.2 LDA function

```
function [W, VsLDA, DsLDA, colors] = newLDA (c1, c2, nLD)
% Number of observations in each class
n1 = size(c1,2);
n2 = size(c2,2);

% Mean of each class
mu1=mean(c1,2);
mu2=mean(c2,2);

% Label vector
one_vec = ones(n1,1);
two_vec = -ones(n2,1);
colors = [one_vec; two_vec];

% Calculate the between class variance (SB)
Sb = (mu1-mu2)*(mu1-mu2)';  % eq: 3.7
d1=c1-repmat(mu1,1,size(c1,2));
d2=c2-repmat(mu2,1,size(c2,2));

% Calculate the within class variance (SW)
s1=d1*d1';
s2=d2*d2';
Sw = s1 + s2;                % eq: 3.9

%% Find eigen values and eigen vectors of J(w)
[evecLDA,evalLDA]=eig(Sb,Sw); % eq: 3.10

%% LDA Step 3: Sorting and finding the highest eigenvalues -->
    eigenvectors (2)
[dLDA,indLDA] = sort(diag(evalLDA),'descend');
DsLDA = evalLDA(indLDA,indLDA);
VsLDA = evecLDA(:,indLDA);

%% Projecting the data into LDA1 and LDA2
w = [VsLDA(:,1) VsLDA(:,2) VsLDA(:,3) VsLDA(:,4) VsLDA(:,5) VsLDA(:,6)
    ];
W = w(:,1:nLD);
```

Source A.2: The LDA function used for dimensionality reduction

A.3 SVM function

```
function [w, b, Sv] = SVM (c1t, c2t, c1v, c2v, possible_C)
% Concatenating class 1 and 2
Data_train = [c1t c2t];
Data_test = [c1v c2v];

% Label vector for train and test data
Label_train = [ones(1,size(c1t,2)) (-ones(1,size(c2t,2)))];
Label_test = [ones(1,size(c1v,2)) (-ones(1,size(c2v,2)))];

% Number of observations in Data_train
n_SVM = size(Data_train,2);

% Position in the vector for class 1 and 2
Label = Label_train;
ClassA = find( Label == +1 );
ClassB = find( Label == -1 );

% Design SVM
% Quadratic objective term
H = zeros(n_SVM,n_SVM);
for i = 1:n_SVM
    for j = i:n_SVM
        H(i,j) = Label_train(i)*Label_train(j)*Data_train(:,i)'*
    Data_train(:,j); % Saves the value to the next column
        H(j,i) = H(i,j); % Saves the value to the next row
    end
end

% Linear objective term
f = -ones(n_SVM,1);

% Linear equality constaints
Aeq = Label_train;
beq = 0;

% Lower bounds
lb=zeros(n_SVM,1);

% Optimization options
Alg{1} = 'trust-region-reflective';
Alg{2} = 'interior-point-convex';
options = optimset('Algorithm', Alg{2}, 'Display', 'off', 'MaxIter',20)
    ;

% Saving the accuracy for all possible_C
Acc_matrix = [size(possible_C,2),2];

% Finding the optimal C for all possible C
for j= 1:size(possible_C,2)
    C = possible_C(j);
```

```
49
50      % Upper bounds
51      ub=C*ones(n_SVM,1);
52
53      % Finding alpha (lagrangian multipliers)
54      alpha = quadprog(H,f,[],[],Aeq,beq,lb,ub,[],options)';
55      AlmostZero = (abs(alpha)<max(abs(alpha))/1e5);
56      alpha(AlmostZero) = 0;
57
58      % Determining the support vectors
59      Sv = find(alpha>0 & alpha<C);
60
61      % Calculating the weight vector
62      w = 0;
63      for i = Sv
64          w = w+alpha(i)*Label_train(i)*Data_train(:,i); % eq: 3.21
65      end
66
67      % Calculating the bias
68      b = mean(Label_train(Sv)-w'*Data_train(:,Sv)); % eq: 3.22
69
70      % Predicting the class for the test data
71      Predict = sign(Data_test'*w+b); % eq: 3.18
72
73      % Calculating the accuracy based on the prediction and known labels
74      NPredict = size(Predict,1);
75      LP_compare = Label_test' == Predict;
76      accuracy = (sum(LP_compare) / NPredict)*100;
77
78      % Saving the accuracy to a vector
79      Acc_matrix(j,:) = [C accuracy];
80
81      % Displaying the current C value test
82      Text = ['Finished testing C = ',num2str(C)];
83      disp(Text)
84  end
85
86  % Rerun the test using the optimal C value
87  % Choosing the C value with the most accuracy
88  [max_C, indx] = max(Acc_matrix(:,2));
89  c_optimal = Acc_matrix(indx,1);
90
91  % Upper bounds
92  ub = c_optimal *ones(n_SVM,1);
93
94  % Calculating alpha
95  alpha = quadprog(H,f,[],[],Aeq,beq,lb,ub,[],options)';
96  AlmostZero = (abs(alpha)<max(abs(alpha))/1e5);
97  alpha(AlmostZero) = 0;
98
99  % Finding the support vectors
100 Sv = find(alpha>0 & alpha<c_optimal );
```

```
101
102 % Calculating the weight vector
103 w = 0;
104 for i = Sv
105     w = w+alpha(i)*Label_train(i)*Data_train(:,i); % eq: 3.21
106 end
107
108 % Calculating the bias
109 b=mean(Label_train(Sv)-w'*Data_train(:,Sv)); % eq: 3.22
110
111 % Predicting the class for the test data
112 Predict = sign(Data_test'*w+b); % eq: 3.18
113
114 % Calculating the accuracy based on the prediction and known labels
115 NPredict = size(Predict,1);
116 LP_compare = Label_test' == Predict;
117 accuracy = (sum(LP_compare) / NPredict)*100;
```

Source A.3: The SVM function used for classification

APPENDIX B

Newly simulated data

In the table below a list of all the newly simulated test data are shown with the specified T_{set}, $HeatLoad$ and $Offset/Scale$ for the different fault signals. The $Offset$ and $Scale$ values for the tests have been changed in order to simulate more realistic data. Commonly for all the tests the T_{amb} is set at 25 $°C$ while the fault sample has been set to start after sample 3001 reducing the sample size to 6000.

Table B.1: List of the newly simulated data

Simulation No.	T_{set}	$HeatLoad$	Fault Signal
1	0	1000	Tsuc Offset (-2)
2	0	13000	Tsuc Offset (-2)
3	7	2000	Tsuc Offset (-2)
4	7	17000	Tsuc Offset (-2)
5	12	4000	Tsuc Offset (-2)
6	12	20000	Tsuc Offset (-2)
7	0	1000	Tsup Offset (-2)
8	0	13000	Tsup Offset (-2)
9	7	2000	Tsup Offset (-2)
10	7	17000	Tsup Offset (-2)
11	12	4000	Tsup Offset (-2)
12	12	20000	Tsup Offset (-2)
13	0	1000	Tret Offset (-2)
14	0	13000	Tret Offset (-2)
15	7	2000	Tret Offset (-2)
16	7	17000	Tret Offset (-2)
17	12	4000	Tret Offset (-2)
18	12	20000	Tret Offset (-2)

19	0	1000	Tdis Offset (-2)
20	0	13000	Tdis Offset (-2)
21	7	2000	Tdis Offset (-2)
22	7	17000	Tdis Offset (-2)
23	12	4000	Tdis Offset (-2)
24	12	20000	Tdis Offset (-2)
25	0	1000	Pdis Offset (-1)
26	0	13000	Pdis Offset (-1)
27	7	2000	Pdis Offset (-1)
28	7	17000	Pdis Offset (-1)
29	12	4000	Pdis Offset (-1)
30	12	20000	Pdis Offset (-1)
31	0	1000	Psuc Offset (-0.1)
32	0	13000	Psuc Offset (-0.1)
33	7	2000	Psuc Offset (-0.2)
34	7	17000	Psuc Offset (-0.2)
35	12	4000	Psuc Offset (-0.2)
36	12	20000	Psuc Offset (-0.2)
37	0	1000	Cpr Scale (0.8)
38	0	13000	Cpr Scale (0.8)
39	7	2000	Cpr Scale (0.8)
40	7	17000	Cpr Scale (0.8)
41	12	4000	Cpr Scale (0.8)
42	12	20000	Cpr Scale (0.8)
43	0	1000	Exv Scale (0.8)
44	0	13000	Exv Scale (0.8)
45	7	2000	Exv Scale (0.8)
46	7	17000	Exv Scale (0.8)
47	12	4000	Exv Scale (0.8)
48	12	20000	Exv Scale (0.8)
49	0	1000	Evap Fan Scale (0.8)
50	0	13000	Evap Fan Scale (0.8)
51	7	2000	Evap Fan Scale (0.8)
52	7	17000	Evap Fan Scale (0.8)
53	12	4000	Evap Fan Scale (0.8)
54	12	20000	Evap Fan Scale (0.8)
55	0	1000	Cond Fan Scale (0.8)
56	0	13000	Cond Fan Scale (0.8)
57	7	2000	Cond Fan Scale (0.8)
58	7	17000	Cond Fan Scale (0.8)
59	12	4000	Cond Fan Scale (0.8)
60	12	20000	Cond Fan Scale (0.8)
61	0	1000	Tsuc Offset (2)

62	0	13000	Tsuc Offset (2)
63	7	2000	Tsuc Offset (2)
64	7	17000	Tsuc Offset (2)
65	12	4000	Tsuc Offset (2)
66	12	20000	Tsuc Offset (2)
67	0	1000	Tsup Offset (2)
68	0	13000	Tsup Offset (2)
69	7	2000	Tsup Offset (2)
70	7	17000	Tsup Offset (2)
71	12	4000	Tsup Offset (2)
72	12	20000	Tsup Offset (2)
73	0	1000	Tret Offset (2)
74	0	13000	Tret Offset (2)
75	7	2000	Tret Offset (2)
76	7	17000	Tret Offset (2)
77	12	4000	Tret Offset (2)
78	12	20000	Tret Offset (2)
79	0	1000	Tdis Offset (2)
80	0	13000	Tdis Offset (2)
81	7	2000	Tdis Offset (2)
82	7	17000	Tdis Offset (2)
83	12	4000	Tdis Offset (2)
84	12	20000	Tdis Offset (2)
85	0	1000	Pdis Offset (1)
86	0	13000	Pdis Offset (1)
87	7	2000	Pdis Offset (1)
88	7	17000	Pdis Offset (1)
89	12	4000	Pdis Offset (1)
90	12	20000	Pdis Offset (1)
91	0	1000	Psuc offset (0.1)
92	0	13000	Psuc offset (0.1)
93	7	2000	Psuc offset (0.2)
94	7	17000	Psuc offset (0.2)
95	12	4000	Psuc offset (0.2)
96	12	20000	Psuc offset (0.2)
97	0	1000	Cpr Scale (0.2)
98	0	13000	Cpr Scale (0.2)
99	7	2000	Cpr Scale (0.2)
100	7	17000	Cpr Scale (0.2)
101	12	4000	Cpr Scale (0.2)
102	12	20000	Cpr Scale (0.2)
103	0	1000	Evx Scale (0.2)
104	0	13000	Exv Scale (0.2)

105	7	2000	Exv Scale (0.2)
106	7	17000	Exv Scale (0.2)
107	12	4000	Exv Scale (0.2)
108	12	20000	Exv Scale (0.2)
109	0	1000	Evap Fan Scale (0.2)
110	0	13000	Evap Fan Scale (0.2)
111	7	2000	Evap Fan Scale (0.2)
112	7	17000	Evap Fan Scale (0.2)
113	12	4000	Evap Fan Scale (0.2)
114	12	20000	Evap Fan Scale (0.2)
115	0	1000	Cond Fan Scale (0.2)
116	0	13000	Cond Fan Scale (0.2)
117	7	2000	Cond Fan Scale (0.2)
118	7	17000	Cond Fan Scale (0.2)
119	12	4000	Cond Fan Scale (0.2)
120	12	20000	Cond Fan Scale (0.2)