FAULT DIAGNOSING OF MULTIVARIATE PROCESSES BASED ON DATA MINING

by

Yildiz Terkesli
B.Sc., Computer Engineering, Istanbul Technical University, 1998

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Summer 2006

APPROVAL

Name:	**Yildiz Terkesli**
Degree:	**Master of Applied Science**
Title of Thesis:	**Fault Diagnosing of Multivariate Processes Based on Data Mining**
Examining Committee:	
Chair:	**Dr. Ivan V. Bajic** Assistant Professor, School of Engineering Science
	Dr. Mehrdad Saif Senior Supervisor Professor, School of Engineering Science
	Dr. John D. Jones Supervisor Associate Professor, School of Engineering Science
	Dr. William A. Gruver **Internal Examiner** Professor, School of Engineering Science
Date Approved:	August 1, 2006

ABSTRACT

In modern industrial plants, large numbers of process measurements are stored in historical databases providing a potentially valuable source of process information. One potential use for historical plant data is as an aid in fault diagnosis. However, information contained in these databases has been underutilized for several reasons. First, the volume of data that must be analyzed is enormous. Second, the data are multidimensional. Third, the variables are interrelated and need to be considered simultaneously in the analysis.

In this thesis, a new data mining framework combining principal component analysis (PCA) and modern data mining techniques (k-Means clustering and decision tree induction techniques) is developed to exploit multivariate process data to detect and identify process faults.

An extensive simulation study for a three-tank benchmark system demonstrates that this strategy is more effective than existing PCA methods in detecting system faults. It can also successfully distinguish between 20 different system faults.

Keywords:

Fault detection; fault identification; principal component analysis; data mining

To my parents.

ACKNOWLEDGEMENTS

I would like to thank all of the friends, family and staffs of School of Engineering Science who have, in various ways, helped me complete this thesis. First, I would like to thank my senior supervisor Professor Mehrdad Saif, for his guidance and insightful advice in completing my thesis.

I would also like to express my gratitude to Dr. John Jones for serving as my Supervisory Committee member and Dr. William Gruver for accepting to be my internal examiner.

I would also like to thank Department of Computer Automation and Control at Jozef Stefan Institute for their permission to use the three-tank benchmark system in this research. I give special thanks to Dr. Damir Vrancic of Department of Computer Automation and Control at Jozef Stefan Institute for his extra effort in providing me Simulink files of the benchmark system.

Most importantly, I would like to thank my family for supporting me during these years. Regardless of how far away they are, this research couldn't have been accomplished without their constant support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE OF ACRONYMS

AEM:	Abnormal Event Management
ART:	Adaptive Resonance Theory
DM:	Data Mining
FDI:	Fault Detection and Isolation
FFNN:	Feedforward Neural Networks
KDD:	Knowledge Discovery in Databases
MSPM:	Multivariate Statistical Process Monitoring
NLPCA:	Nonlinear Principal Component Analysis
PCA:	Principal Component Analysis
PC:	Principal Component
SPE_x :	Squared Prediction Error

CHAPTER 1: INTRODUCTION

Fault detection and diagnosis is an important problem in process engineering. Early detection and diagnosis of process faults while plant is still operating in a controllable region can help avoid catastrophic failures and reduce productivity loss. To improve the efficiency, reliability and safety of modern complex systems, advanced methods of supervision, fault detection and fault diagnosis have become increasingly important for many technical processes. Since the petrochemical industries lose an estimated 20 billion dollars every year, they rated abnormal event management (AEM) as their number one problem that needs to be solved [36].

The discipline of process control has made tremendous progress in the last three decades with advances in the computer control of complex systems. However, process fault detection and diagnosis, a very important control task in managing process plants, still remains largely a manual activity, performed by the human operators. This task has become increasingly difficult due to the broad scope of the diagnostic activities and size and complexity of modern process plants. For example, in large scale process plant there may be as many as 1500 process variables observed every few seconds leading to a massive amount of historical data [37]. Furthermore, the task of fault diagnosis is made difficult by the fact that the process measurements may often be insufficient, incomplete and unreliable due to a variety of causes such as sensor biases or failures. Thus, the current challenge for control engineers is the automation of process fault detection and diagnosis.

Fault diagnosis methods can be broadly classified into three general categories: quantitative model based methods, qualitative model based methods, and process history based methods. The information needed to effectively detect and diagnose fault situations is based on knowledge of the process and analysis of the process data. Two of the main components in a diagnosis classifier are: (1) the type of knowledge and (2) the type of diagnostic search strategy. Diagnostic search strategy is usually a function of knowledge representation scheme and is influenced by the type of a priori knowledge available. Hence, type of a priori knowledge used is the most important distinguishing feature in diagnostic systems. The basic a priori knowledge that is needed for fault diagnosis is the set of failures and the relationship between the observations and the failures. A diagnostic system may have them explicitly (as in a table lookup), or it may be inferred from some source of domain knowledge. The a priori domain knowledge may be developed some fundamental understanding of the physics of the process, which is referred as model-based knowledge. On the other hand, it may be gleaned from past experience with process, which is called process history based knowledge. In contrast to the model-based approaches (either quantitative model based or qualitative model based) where a priori knowledge about the model of the process is assumed, in process history based methods only the availability of large amount of historical process data is assumed [36].

In process operation and control, modern distributed control and modern data logging systems allow enormous amounts of data which contain valuable information about both normal and abnormal operations to be routinely collected and stored. There is also product quality, production, and maintenance data which are stored less frequently. Thus, a massive amount of process data (past and present) is available for analysis. Despite industrial interest and significant potential benefits of the historical plant data, the information contained in these databases has remained elusive due to the challenges encountered trying to extract it. It is a well-known fact that industrial plants are “data rich, but information poor”. The largest obstacle to using historical data is simply finding relevant patterns in such a vast sea of data. In a typical large scale industrial plant there can be thousands of measured variables with measurements being made as frequently as every fraction of a second. In order to use such a large database as an aid in fault diagnosis, proper techniques must be used which require minimal computer time while still revealing the unique characteristics of the process data.

Multivariate statistical techniques such as principal component analysis (PCA) have received a great deal of attention in recent years for their ability to successfully determine when a fault has occurred. However, diagnosing the fault has proven to be much more difficult and often requires process knowledge (e.g. expert systems) or a set of reference data for each possible fault (pattern recognition or supervised learning).

In recent years, there have been significant developments in extracting information from large databases and in automating data analysis. A research community has been developed under the label of data mining and knowledge discovery in databases (KDD). The goal of data mining is to discover previously unknown but potentially useful patterns or relationships in a database. Data mining techniques have been successfully applied to databases in space, telecommunication, business, and marketing industries. However, there appear to be very few, if any, applications of data mining to fault detection and isolation problems.

The major challenge in developing a fault diagnosis system arises from the characteristics of operational data, which are summarized as follows:

- Large volume: Large volumes of data demand large computer memory and high speed.

- High dimensionality: The behavior of a process is usually defined by a large number of correlated variables. Dimension reduction is required to visualize the process behavior.

- Process uncertainty and noise: Good data pre-processing techniques are required to clean the data.

- Process dynamics: Many data mining and knowledge discovery tools are mainly designed to handle categorical values such as temperature being high or low. They are not effective in dealing with continuous-valued variables. It is very important to design tools or techniques that are able to handle variables that take values as dynamic trends

- Complex interactions between process variables: Many techniques require attributes to be independent. However, many process variables are interrelated and therefore need to be considered simultaneously in the analysis.

There has been a significant progress in automating data analysis for process monitoring and fault diagnosis by successful applications of the machine learning techniques in data mining process. These methods can be roughly divided into two categories: supervised and unsupervised [2]. Supervised techniques are associated with assignment of a set of unknown data to previously known classes according to a similarity measure. Supervised methods need a large number of data sets with known classes as training data to train the models. A typical example would be the feedforward neural network (FFNN). Though they can generally give accurate results, supervised methods are not applicable when training data are not available. Unsupervised approaches, which can learn from unknown to predict unknown, can be used in this situation. Widely studied unsupervised learning methods include nonlinear principal component analysis, adaptive resonance theory (ART), and Bayesian automatic classifications. However, the main limitation of supervised and unsupervised approaches mentioned above is that they give predictions but are not able to give causal explanations about the root cause of the fault.

Besides the ability to identify the source of malfunction, a diagnostic system should also provide explanations on how the fault originated and propagated to the current situation. This is a very important factor in designing on-line decision support system. This requires the ability to reason about the cause and effect relationships in a process. A diagnostic system has to justify its recommendations so that the operator can accordingly evaluate and act using his/her experience [36].

The objective of this research project is to develop an integrated data mining framework for fault detection and isolation (FDI) using historical process data. It combines the multivariate statistical process monitoring (SPM) technique with modern data mining methods. The approach presented in this research is able to not only detect and isolate faults using training data set, but also generate causal knowledge indicating the variables that are responsible for the malfunction. The approach described in this thesis utilizes a technique developed by Wang and Li (1999), which uses PCA to process dynamic time series data. Then, an unsupervised learning technique, *k*-Means clustering, is used to conceptualize the process data for subsequent supervised learning, decision tree classification. Here, decision trees are employed to detect and identify various faults of a multivariate process.

The remainder of the thesis is organized as follows: A brief introduction to data mining and knowledge discovery in databases is provided in Chapter 2. The proposed data mining framework is also introduced in Chapter 2. Principal component analysis (PCA), concept formation from dynamic trend signals using PCA and *k*-Means clustering technique are provided in Chapter 3. The decision tree classification algorithm used in this research is introduced briefly in Chapter 4. The performance of the developed framework is compared with the conventional monitoring technique in a detailed simulation study. The implementation of the data mining framework and the results of the simulation are presented in Chapter 5. Finally, the research is summarized along with recommendations for future research in Chapter 6.

CHAPTER 2: AN OVERVIEW OF DATA MINING AND KNOWLEDGE DISCOVERY

2.1 Definition of Knowledge Discovery in Databases

The fast development and widespread application of information and database technologies has created many new opportunities for those working on engineering, science, and business. In the last decade or so, there has been an explosive growth in our capabilities to both generate and collect data. Advances in data storage technologies and database management systems have allowed us to create mountains of stored data. The field of data mining (DM) and knowledge discovery in databases (KDD) has emerged as a new discipline in engineering and computer science due to advances in data collecting technologies and high speed computing.

With the increasing use of databases the need to be able to exploit large volumes of data being generated has become very critical to be competitive. The large volume of data and high dimensionality of databases have made traditional manual methods of data analysis a very difficult and time-consuming task if not impossible. A significant need exists for a new generation of techniques and tools with the ability to intelligently and automatically assist humans in analyzing the mountains of data for nuggets of useful knowledge. The aim of data mining and KDD is to develop tools and methodologies to automate data analysis process and find useful information and knowledge from data to help in decision-making process.

A widely accepted definition of KDD is given by Fayyad et. al. [3] as:

"the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data."

Fayyad et. al. (1996) also define the process of *data mining* in this context as:

"a step in the KDD process by which patterns are extracted and enumerated from the data."

The analysis of these definitions shows that KDD is a very complicated process comprising a number of steps, and data mining is one step in the process. Since data mining is the central theme to the process of knowledge discovery, these terms are often used interchangeably in the literature. The key aspects of this definition are that the whole KDD process discovers knowledge in the form of patterns from the existing data. These patterns should be understandable and potentially useful to the organizations so that decision makers or domain experts are able to understand the knowledge and use it. Data mining is a process concerned with uncovering meaningful patterns, association, anomalies and statistically significant structures in the data.

Data mining generally refers to the case where the data is too large or too complex and heterogeneous in content to allow either manual analysis or analysis by means of simple tools and queries. In general, one can summarize that for a typical data mining case [14]:

- The data set can be quite large,
- The problem generally challenging and is often not well defined,
- There are missing and faulty data,
- There are redundancies in the data fields, but the redundant fields do not all have the same quality.

Data mining is an interdisciplinary science whose domain area ranges from statistics to data warehousing, database systems, pattern recognition, machine learning, artificial intelligence, high performance computing, signal and image processing, computer visualization, etc. As data mining has been applied to new problem domains, this technology mix has grown as well. Data mining and KDD borrows methods, algorithms, and technologies from these diverse fields to extract knowledge from data, develop means of classifying the data, and discover patterns in the data. The idea behind it is to look at data in a different way and to use innovative and effective data visualization techniques so that we can obtain a novel or deeper understanding of it.

Data mining and KDD are potentially valuable in virtually every industrial and business sector where database and information technology is used. Data mining techniques are being applied for the analysis of data in a variety of fields including medical imaging, astronomy, bio-informatics, web mining, text mining, customer relationship management, market-basket analysis, fraud detection, portfolio trading, manufacturing process analysis, experiment result analysis and scientific data analysis. Recently, some researchers have applied data mining techniques in control engineering for fault detection and identification problem. [15, 16]. Scientific data mining distinguishes itself in the sense that the nature of datasets is often very different from traditional market-driven data mining applications. The datasets now might involve vast amounts of continuous data, and accounting for underlying system nonlinearities can be extremely challenging from a machine learning point of view [14].

2.2 The KDD Process

The KDD method is interactive and iterative, involving numerous steps with many decisions made by the user. When it is applied to real world problems, these tasks can be very complex. A typical KDD process can be broken down into the following steps [3]:

1. Developing an understanding of the application domain, the relevant prior knowledge and the goals of the end-user.

2. Creating a target data set: selecting a data set or focusing on a subset of variables or real world data on which discovery is to be performed.

3. Data pre-processing and cleaning: basic operations such as the removal of noise or outliners if appropriate, collecting the necessary information to model and deciding on strategies for handling missing data fields, and accounting for time sequence information and known changes.

4. Data reduction and projection: finding useful features to represent the data depending on the goal of the task. Using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representation for the data.

5. Choosing the data mining task: deciding whether the goal of the KDD process is logical, summarizing, classification, regression, prediction, and clustering etc.

6. Choosing the data analysis algorithms: selecting methods to be used for searching for patterns in the pre-processed data. This includes deciding which models and parameters may be appropriate and matching a particular data mining method with overall criteria of KDD. Most of the time, the person who runs the data mining task should investigate several analysis models before being able to choose one of them. After method selection, the analyst has to select the important parameters of the model.

7. Data mining: searching for patterns of interest in a particular representational form or a set of such representation, including classification rules or trees, regression, clustering, sequence modeling, dependency, and so forth. The user can significantly aid the data mining method by correctly performing the preceding steps.

8. Interpreting mined patterns, and possible return to any of the previous steps.

9. Consolidating the discovered knowledge: incorporating this discovery knowledge into the performance system, taking actions based on the knowledge, and reporting it to interested parties. This also includes checking for and resolving potential conflicts with previously believed (or extracted) knowledge.

The KDD process can involve significant iterations and may contain loops between any two steps. The basic flow of steps is illustrated in Figure 2.1.

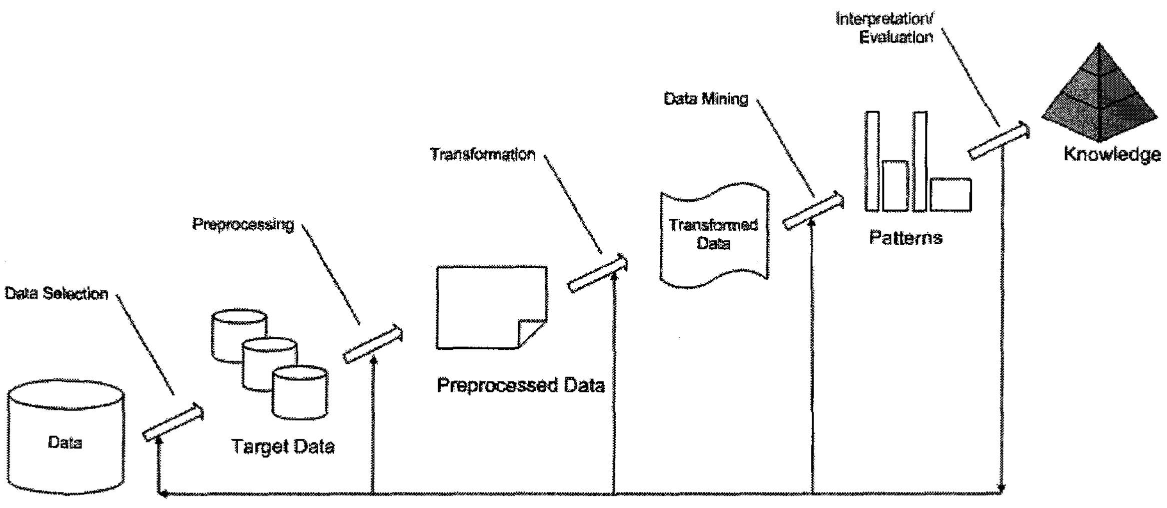


Figure 2.1 An overview of the steps comprising the KDD process

During the data pre-processing, cleaning and data reduction steps, relevant high level features or attributes are extracted from the low level data. To ensure the success of the data mining process, it is important that the features extracted from data are relevant to the problem and representative of the data. Although much of the focus in KDD process is on the data mining step, the other steps are of considerable importance for the successful application of KDD in practice. It is important to note that often a large amount of effort is required before the data can be presented in a format that is suitable for data mining. Data cleaning and pre-processing often takes up a large part of the resources committed to a typical data mining project and might involve 80 percent of the effort. It is often necessary to experiment with different data transformations and dimension reduction techniques (e.g. Fourier and wavelet transforms, PCA) in the data pre-processing step.

2.3 Data Mining Methods and Techniques

A wide variety of methods and techniques are commonly used in data mining applications. In general, data mining tasks can be classified into two categories: descriptive and predictive [4]. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks perform inference on the current data in order to make predictions.

According to functions and application purposes, data mining methods can involve clustering, classification, attribute and feature selection, the formation of rules and outlier detection. These techniques can be based on statistics, probability theory, Bayesian networks, decision trees, association rules, neural networks, evolutionary computation, and fuzzy logic [1, 4]. A very brief review of the techniques that is used in this research is given in the following section.

2.3.1 Clustering

Clustering, which is also called unsupervised machine learning, aims to generate a classification scheme for grouping the objects into a number of classes such that objects within a class are similar, in some respect, but distinct from those from other classes. This involves determining both the number and description of the classes. Unlike classification, which analyzes class-labeled data objects, clustering analyzes data objects without consulting a known class label. In general, class labels are not present in the training data because they are simply not known.

The grouping often depends on calculating a similarity or distance measure. The objects are clustered or grouped based on the principle of maximizing the in-class similarity and minimizing the interclass similarity. Clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Clustering is a useful step to look at the data before further analysis is carried out. The data mining methods to be applied can be further defined based on the prior knowledge from data gathered through clustering. Grouping multivariate data into clusters according to similarity or dissimilarity measures is the goal of some applications. Examples of the clustering methods are unsupervised neural networks, including self-organizing Kohonen neural networks, Bayesian automatic classification, partitioning methods (*k*-Means and *k*-Medoids).

2.3.2 Classification

Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown [4]. Classification is also called supervised machine learning because it always requires data patterns with known class assignments to train a model which is then used for predicting the class assignments of new data patterns. The derived model is based on the analysis of a set of training data. The derived model may be represented in various forms, such as classification rules (IF-THEN), decision trees, mathematical formulas, or neural networks. A decision tree is a flow chart like tree structure, where each node denotes an attribute value, each branch represents an outcome of the test, and tree leaves represent classes.

2.3.3 Conceptual Clustering and Classification

Most clustering and classification algorithms depend on numerically calculating some sort of similarity or distance measure, and because of this they are often called similarity based methods. On the other hand, conceptual clustering and classification develops a qualitative language for describing the knowledge used for clustering and is basically in the form of decision trees or production rules. The inductive system C4.5 is a typical approach [26, 27], which is able to automatically generate decision trees and production rules from the data, which is pre-processed and converted into flat file format. Decision trees and rules have a simple representation, making the derived model easy to comprehend by the end user. However, available approaches were mainly developed for problem domains in which variables take only categorical values, such as temperature being high and low. They are not effective in dealing with time series or numerical data. Discretization of numerical time series variables to categorical values is necessary to successfully apply this technique to real world engineering domains.

2.3.4 Mining Time Series Data

Data mining of historical databases for process monitoring and fault diagnosis has started to receive attention in the computer science literature; however problems involving time-series databases have been addressed only recently. Many industrial and business areas deal with time-series or dynamic data. All statistical and real-time control data in today's process monitoring and control systems are essentially time-series [1]. However, to make use of continuous process data in a computer system, it is required to compress the dynamic data in order to reduce dimensions.

It is very easy for humans to capture features of each dynamic trend and identify their differences. However, it is very difficult for computers to do the same task. Most KDD techniques cannot account for the time series data. To make the time-series data ready for a data mining task, one has to carry out pre-processing of the data to use minimum data points to capture the features and remove noise. Some of the techniques that have been used to pre-process the dynamic trend signals are Kalman filters, Fourier and wavelet transforms, and multivariate statistical techniques like PCA and neural networks.

2.4 Problem Description

Plant operators have long recognized the value of historical process data and have collected vast amount of data using advanced data collection and storage systems. In the process industries, interest in collecting and storing process data has increased to the point that majority of modern industrial plants use commercial data historians to collect and store process measurements in a historical database [6]. It would be beneficial if these data could be categorized into groups of operating conditions so that the characteristics of these groups can be used for decision support in fault detection and diagnosis (Wang and McGreavy, 1998).

Despite the significant potential benefits of historical process data, it has remained very difficult to extract the information contained in these databases due to a number of reasons. First the data volume is too large and data is multidimensional in nature. Second, the variables recorded in a plant are highly correlated and therefore need to be processed simultaneously in analysis. Other factors that make data processing a very challenging task are noise, uncertainty and dynamics of the system (e.g. nonlinearity).

Recently there has been a significant progress in applying data analysis methods for process monitoring and fault diagnosis. These methods can be roughly divided into two categories: supervised and unsupervised [1]. Supervised methods need a large number of data sets with known classes to train the models. FFNN and decision trees are well known supervised machine learning techniques. Although supervised methods can give accurate results, they are not applicable for the domains for which training data are not available. Unsupervised approaches can learn from grouping data sets into classes based on a distance or similarity measure. Unsupervised learning methods which have been studied for operational state identification and fault diagnosis are nonlinear principal component analysis, adaptive resonance theory (ART), and Bayesian automatic classification.

All supervised and unsupervised approaches mentioned above use the notion of similarity to build their models. A major limitation of distance-based clustering is that it gives predictions but not casual and qualitative explanations. This means that for a process monitoring system it is not able to provide any clues of what variables are responsible for the observed fault.

2.5 Proposed Data Mining Framework

In this research project, a combined data mining framework is proposed for detection and identification of faults for multivariate nonlinear systems. Proposed system is able to project the operation of the process over a specific period of time to a point in the two-dimensional principal component space identifying fault, and generate casual knowledge indicating the variables that are responsible for the abnormal situation. If the same type of fault has occurred in the past, then the relevant historical data provide a valuable source to detect and identify future faults. The ability to give causal explanations, which is easy to understand by plant operators, will be advantageous for difficult process diagnosis problems.

A number of methodologies can be used to create sophisticated fault detection and identification models depending on various data mining techniques. The use of artificial neural networks creates a complex model that is very accurate in terms of predictions and learning the nature of the data sets. However, models created using neural networks are not very easy for humans to comprehend. On the other hand, decision trees offer a mechanism of creating models of the process data that is easy to understand. The proposed system is based on a decision tree, which attempts to build a conceptual language for describing an object, by drawing inductive inference from a training data set. The focus of the algorithm is on deriving rules or decision trees from unordered sets of examples. This attribute-based induction method, a formalism where examples are described in terms of a fixed collection of categorical attributes, differs from other learning methods such as FFNN. Several approaches to inductive learning have been proposed, the most successful one being C4.5 (a successor of ID3 algorithm), a decision tree learning program, developed by Quinlan [26, 27]. However, the decision tree technique is not useful when we want to make predictions for a continuous process variable.

Proposed data mining framework has been implemented using See5 (which is a commercial software package that can run on Windows® platform and has evolved from its early version C4.5) to build a classification model from historical process data. In applying this approach, a critical step is to deal with continuous process variables. Because for fault detection and identification, we need to deal with variables whose values are continuous time series data. One of the major limitations of ID3 was that it assumed that the values of all attributes are discrete. Although C4.5 was claimed to be able to deal with continuous-valued attributes, results are not satisfactory according to many researchers [1, 2].

2.6 Pre-processing Continuous Process Variables

Mining time-series data has attracted great attention as data mining and KDD techniques have been successfully applied to many engineering application. On the other hand not many researches have been done on concept formation from dynamic trend signals. Wang and Li [2] described a methodology for concept formation from time-series data using principal component analysis (PCA). In this approach, the dynamic trends are represented using principal components of the data. The datasets are then projected onto two-dimensional plane for concept formation using the first two principal components of each variable. Their approach relies on the visual examination of this projection to cluster the datasets and requires tremendous user input. Although this is an interesting technique to extract concept from multivariate time-series data, it can become tedious for a large number of process variables.

In this study, a similar methodology to pre-process multivariate time-series data is used by the proposed framework. Rather than visually examining the two-dimensional principal component plots of the process data, *k*-Means clustering is applied to automatically extract concept for subsequent classification process. Proposed data mining framework does not require a priori knowledge about process. The sole purpose of plotting the principal components onto two-dimensional plane is to have an interactive KDD process to fully utilize the data-mining paradigm. Clustering component can also be developed as a batch process without requiring user input, making the framework a viable option for very complex systems. This possibility will be discussed in detail in Chapter 6 when we discuss the future research.

The overall data-mining framework for fault diagnosis is illustrated by the flow chart in Figure 2.2. First, a PCA model of each process variable is built using data sets containing both normal operating periods and a wide variety of abnormal situations or faults. Next, the result is plotted on a two dimensional plane to further cluster process variables by applying *k*-Means clustering techniques. At this point, the framework optionally interacts with the end user to identify the optimum number of cluster for each process variable. Thus, concept formation from process variables whose values are dynamic trend signals has been accomplished. The conceptualized process data can be used in the next step to build a classification model of the plant. Then, test data sets can be fed into the model to detect and identify unknown faults. It is important to note here that the classification model of the system is built iteratively. If the performance of the tree is not satisfactory for the given training data set, end user can go back to the clustering step to further refine the concept formation process.

2.7 Conclusions

The proposed system combines the advantages of both standard multivariate SPM methods and modern data mining techniques for fault diagnosis purpose. Specifically, an integrated data mining scheme sequentially adopting the techniques of PCA, data clustering, and decision trees is developed. Proposed framework is basically a conceptual clustering system based on inductive machine learning approach. Commercially available version of famous C4.5 algorithm is used to build decision trees from historical plant data.

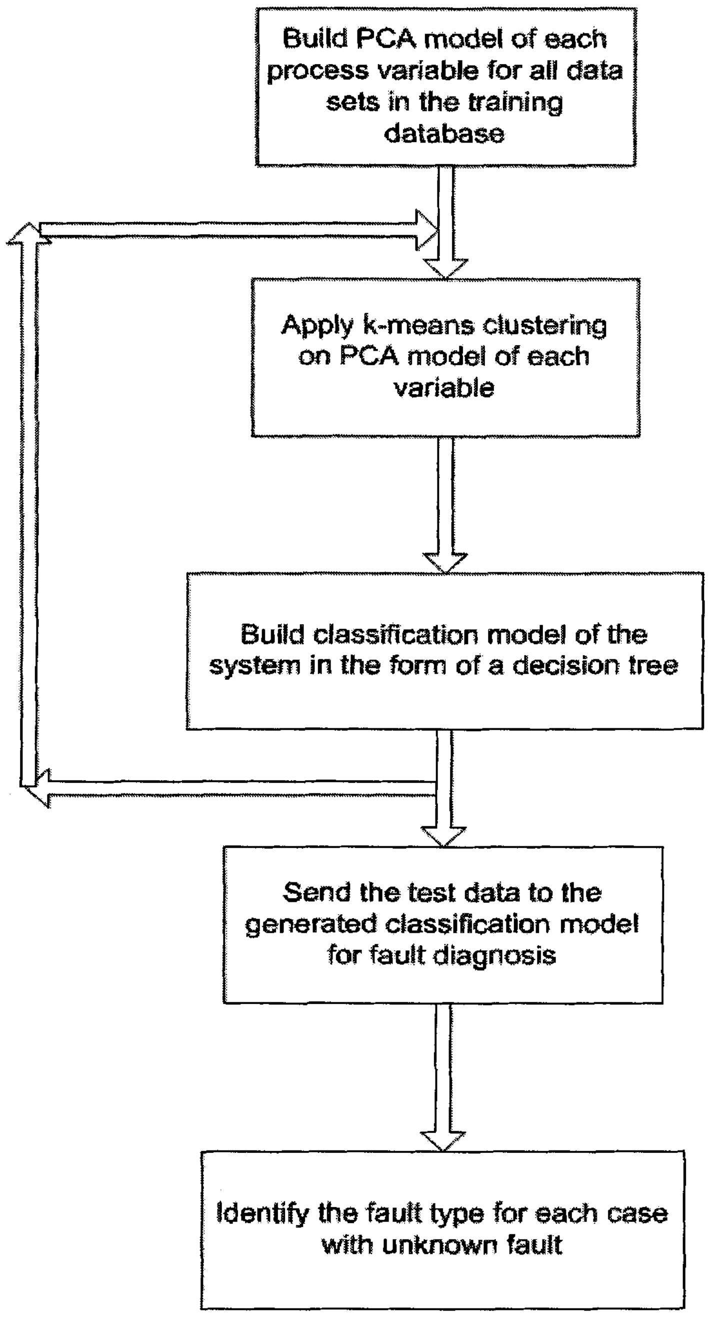

Build PCA model of each process variable for all data sets in the training database
Apply k-means clustering on PCA model of each variable
Build classification model of the system in the form of a decision tree
Send the test data to the generated classification model for fault diagnosis
Identify the fault type for each case with unknown fault

Figure 2.2 Proposed data mining framework for fault detection and identification

The schema uses the data pre-processing approach that combines the PCA based concept formation of continuous process variables proposed by Wang and Li [2] and k-Means clustering, a distance based clustering technique.

The analysis results of this scheme are very easy to comprehend by the end user. It is completely data driven, so it does not require a priori knowledge of the system model or process variables. It predicts the given unknown faults using known data and produce results in the form of decision trees enabling further investigation on faults. The computational load is modest, which allows processing of large amounts of process data in very short time.

CHAPTER 3:
DATA PREPROCESSING FOR CONCEPT FORMATION

3.1 Introduction

This chapter describes data pre-processing for noise removal and concept formation from monitored process measurements. The discussion is concerned with capturing the features of a dynamic trend signal from continuous process. A dynamic trend signal is the visualization of a continuous process variable over a time frame and consists of many sample values. However, in order to make effective use of continuous process signals in a data mining system, it is necessary to compress the data to fewer values by keeping important feature of the signal. Many data mining and KDD tools and algorithms have been developed only for dealing with discrete-valued attributes and not effective in dealing with continuous-valued variables. It is not possible to use variables represented by a trend in the inductive machine learning algorithm used in the study without pre-processing the data.

This chapter first introduces principal component analysis (PCA). Next, the technique for concept formation from dynamic trend signals using PCA is described. Then, the concept formation technique used in this study is introduced together with *k*-Means clustering algorithm.

3.2 Principal Component Analysis

The method of PCA was developed in early 1900's, and has now re-emerged as an important data analysis technique used to describe the multivariate structure of the data [7]. It is a multivariate statistical technique in which a set of correlated variables is transformed into a new set of uncorrelated variables. The central idea is to reduce the dimensionality of a data set consisting a large number of interrelated variables, while retaining as much as possible of the variations present in the original data set [1]. The new uncorrelated variables (principal components - PCs) are linear combinations of the original variables. PCA uses all of the original variables to obtain a smaller set of new variables that can be used in place of original variables. The greater the degree of the correlation between the original variables, the fewer the number of PCs required. Dimension reduction capability of PCA makes it a vital tool in data mining activities.

3.3 Theory of PCA

Given a data matrix $X(m \times n)$ representing *m* observations of each of *n* variables, x_1, x_2, ...x_n, the first principal component, p_1, is given by a linear combination of the *n* variables as

$$p_1 = w_{11}x_1 + w_{12}x_2 + \ldots + w_{1n}x_n \tag{3.1}$$

The coefficients (also called weights), $w_{11}, w_{12}, \ldots w_{1n}$, can be written as a vector W_1,

$$p_1 = W_1^T x \tag{3.2}$$

To find the coefficients defining the first principal component, the elements of W_1 should be chosen to maximize the variance of p_1 subject to the normalization constraint,

$$W_1^T W_1 = 1 \tag{3.3}$$

The variance of the first principal component is then given by

$$Var(p_1) = Var(W_1^T x) = W_1^T S W_1 \tag{3.4}$$

where S is the covariance matrix of $X(m \times n)$. The solution of $W_1 = (w_{11}, w_{12}, \ldots, w_{1n})$ to maximize the variance p_1 is the eigenvector of S corresponding to the largest eigenvalue. Therefore, the problem of calculating p_1 has been reduced to an eigenvalue problem. The eigenvalues of S are roots of the following equation:

$$| S - \lambda I | = 0 \tag{3.5}$$

The calculation of the j^{th} principal component is identical to the calculation of the first except for an additional constraint. Similarly the j^{th} PC is a linear combination of the variables

$$p_j = W_j^T x \tag{3.6}$$

which has the greatest variance subject to the following constraints:

$$W_j^T W_j = 1 \tag{3.7}$$

$$W_j^T W_i = 0 \qquad (i < j)$$

This problem can also be solved through the use of covariance matrix of $X(m \times n)$ and it also reduces to an eigenvalue problem. Therefore, for a $m \times n$ data matrix, X, with n variables and m measurements, the n principal components can be solved via an eigenvector decomposition of the covariance matrix,

$$Sp_i = \lambda_i p_i \qquad \text{for } i=1 \text{ to } n \tag{3.8}$$

The first few eigenvectors are the principal components that can capture most of the variance of the original data while the remaining PCs mainly represent the noise.

A useful property of PCA can be given as [7],

$$Tr(S) = Tr(\lambda) \tag{3.9}$$

that is, the sum of the original variances is equal to the sum of the characteristic roots, which are the eigenvalues. This identity is particularly useful in data analysis because it shows that the characteristic roots, which are the variances of the principal components, may be treated as variance components of original variables. The i^{th} eigenvalue of the covariance matrix, λ_i, corresponds to the variance in the original data that is explained by the i^{th} principal component, p_i.

PCA is scale dependent, so that principal components must be scaled in some meaningful way before PCA analysis. There are two ways of scaling principal components, one by rescaling the original data, and the other by rescaling the characteristic vectors. Since PCs are generally regarded as "artificial" variables, scores having unit variances are quite popular for data analysis and quality control applications [7]. In this study, PCs are scaled to unit variance using the following equation:

$$y_i = \frac{p_i}{\sqrt{\lambda_i}} \tag{3.10}$$

where p_i is the principal components for the i^{th} observation (scores), λ_i is the i^{th} eigenvalue of the covariance matrix.

If the eigenvalues are λ_1, λ_2, ...λ_n, then they can be arranged from the largest to the smallest. If they are ordered to satisfy,

$$\lambda_{i-1} \geq \lambda_i \geq \lambda_{i+1} \tag{3.11}$$

then the first principal component describes the most variance in the data while the n^{th} PC describes the least. Usually a large portion of the variance in the data can be described by the first *k* principal components, where *k* < *n*. Original data can be determined from the PCA model only if all PCs are used in the model. If only *k* < *n* PCs are used, the model will not describe some of the variability in the data, In that case,

$$X = t_1 p_1^T + t_2 p_2^T + \ldots + t_k p_k^T + E \tag{3.12}$$

where t_i is the i^{th} score vector and E is the matrix of residuals or model error – the amount that is unexplained by the PC model. If *k* is properly chosen, E should represent only noise and random errors in the original data. Obviously, the larger *k* is, the better fit of the PCA model; the smaller *k* is, the simpler the model will be [7]. The primary advantage of PCA is its potential ability to represent an *n*-variable data set in *k* < *n* dimensions. The important question is: what is the optimal value of *k*? However, there is no standard convention for determining the number of PCs to retain in a PCA model. Jackson [7] explains many criteria that are in use to determine the optimum *k*. One of the widely adopted rules is based on the amount of explained and unexplained variability. In this approach, characteristic roots (eigenvalues) and vectors are obtained until the amount of unexplained variability, or the residual, has been reduced to a predefined quantity. Individual variance explained by each principal component and cumulative variance can be calculated to help determine the number of PC that should be retained.

3.4 Process Monitoring based on PCA

Process monitoring for the abnormal events using principal component analysis typically involves the monitoring of the Q statistic. Q statistic is a measure of the amount of variation not captured by the PCA model. If a totally new type of special event occurs which was not present in the reference data used to develop the in-control PCA model, then new PCs will appear and the new observations will move off the plane. Such new events can be detected by computing the squared prediction error (SPEx) of the residuals of new observations [33].

The Q-statistic is calculated from the error term, E, in the PCA model equation (3.14), and simply represents the sum of squares of the distance of $x_j - \hat{x}_j$ from the *k*-dimensional space that the PCA model defines. Using the PCA model and the score values for time *j*, the measurement vector x_j can be estimated. The PCA model error or residual, e_j, is the difference between this estimate, $\hat{x}_j$, and the actual measurement vector, x_j, as given in the following equation

$$e_j = x_j - \hat{x}_j \tag{3.13}$$

where

$$\hat{x}_j = \sum_{i=1}^{k} t_i(j) p_i \tag{3.14}$$

The Q-statistic for time *j* is then given by,

$$Q_j = e_j^T e_j \tag{3.15}$$

Jackson (1991) showed that approximate confidence limits (upper control limits) can be calculated for the Q-statistic based on the Chi-squared approximation provided that all eigenvalues of the covariance matrix are known.

Confidence limits are given by equations (3.16) to (3.18):

$$Q_\alpha = \theta_1 \left[\frac{c_\alpha \sqrt{2\theta_2 h_0^2}}{\theta_1} + \frac{\theta_2 h_0 (h_0 - 1)}{\theta_1^2} + 1 \right]^{\frac{1}{h_0}} \tag{3.16}$$

Where $$\theta_j = \sum_{i=k+1}^{n} \lambda_i^j \quad \text{for } j = 1,2,3 \tag{3.17}$$

and $$h_0 = 1 - \frac{2\theta_1\theta_3}{3\theta_2^2} \tag{3.18}$$

In equation (3.18) above, α is the significance level (e.g., α = 0.01 corresponds to 99% significance level); and c_α is the normal deviate cutting off an area of α under the upper tail of the distribution if h_0 is positive and under the lower tail if h_0 is negative. θ_j 's are calculated using characteristic roots that correspond to the PCs retained for modeling. This distribution holds whether or not all of the significant principal components are used for PCA modeling [7]. Letting $\alpha = .05$ and $c_\alpha = 1.645$, the limit for Q-statistic can be calculated for 95% confidence limit. Values of Q-statistic higher than this limit, which is denoted by $Q_{.05}$, are an indication that a data vector cannot be adequately represented by PCs that were retained . The confidence limit for Q statistic can be used to monitor this value to determine when the process has deviated from the normal operating region. When the process is in control, Q-statistic (or squared prediction error) represents unstructured fluctuations (noise) that cannot be accounted for by the model. When an unusual event occurs that results in a change in the covariance structure of the system, it will be detected by a high value of SPE_x. A high value of SPE_x means that the projection model is not valid for that observation. The *Q*-statistic with the confidence limits is a very effective multivariate statistical process monitoring technique which can detect the occurrence of faults that cause the process to move away from the hyperplane defined by the reference model.

Correctly scaling the data is also very important in PCA based process monitoring. Some variables exhibit more variability during the course of normal operation, and scaling can prevent these variables from dominating the principal component model. There are two ways of scaling principal components, one by rescaling the original variables, and the other by rescaling the characteristic vectors [7]. In this study, the second approach has been employed by scaling the PCs to have unit variances.

3.5 Data Pre-processing using PCA

In today's modern computer control systems, nearly all important process variables are recorded as dynamic trends. Dynamic trends can be more important than actual real time values in evaluating the current operational status of a continuous process. To make effective use of trends in the subsequent data mining process, it is required to compress the dynamic trend data and to use reduced dimensions to represent the trend features. PCA for data pre-processing is used to serve the following purposes: concept extraction for subsequent data mining step, noise removal, and dimension reduction. Wang and Li [2] proposed a concept formation technique that uses dynamic trend signals of a continuous process variable from all data sets to build the PCA model and then plots the first two PCs in a two dimensional plane. The process data were organized as a $m \times n$ matrix of m observations of n variables. The PCA model of the data was created by unfolding the multivariate dataset into a long row vector and then using the individual elements as features. Then, the datasets were grouped by visually examining the two-dimensional plot of the first two PCs of each variable.

To illustrate this approach, a nonlinear multivariate dynamic system (three-tank benchmark system) was run under different faulty conditions and dynamic trends of all process variables were recorded. Detailed description of the benchmark system and application of the data-mining framework for fault detection and identification can be found in Chapter 5. Here, dynamic trend signals from only one variable are being considered to illustrate the dimension compression capability of PCA. For each data case, thirteen variables are recorded as dynamic responses after a disturbance or fault occurs. Each trend consists of 1200 sample points. Figure 3.1 shows the trends of a variable for two different data cases each representing a different faulty condition.

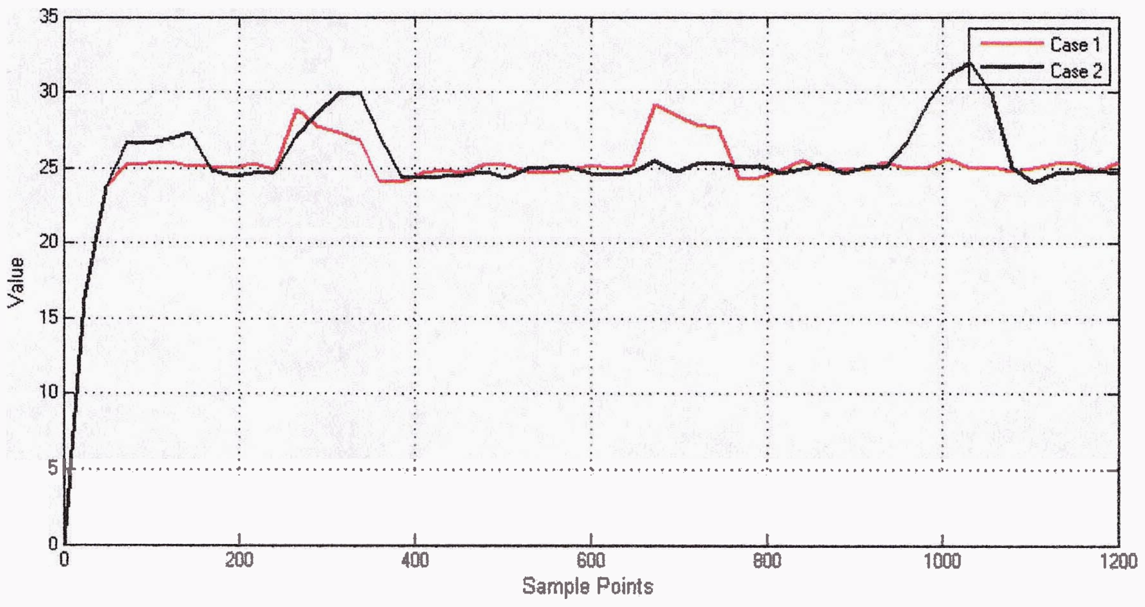


Figure 3.1 Dynamic trends of a variable for two different data cases

The eigenvalues of all 13 principal components for the sample process variable are summarized in Figure 3.2 by plotting the values of the roots (eigenvalues) versus the root number (the principal component number). Since the covariance matrix formed from the data set containing 13 process variables from the benchmark system, we will have 13 eigenvalues whose sum will total to the number of process variables.

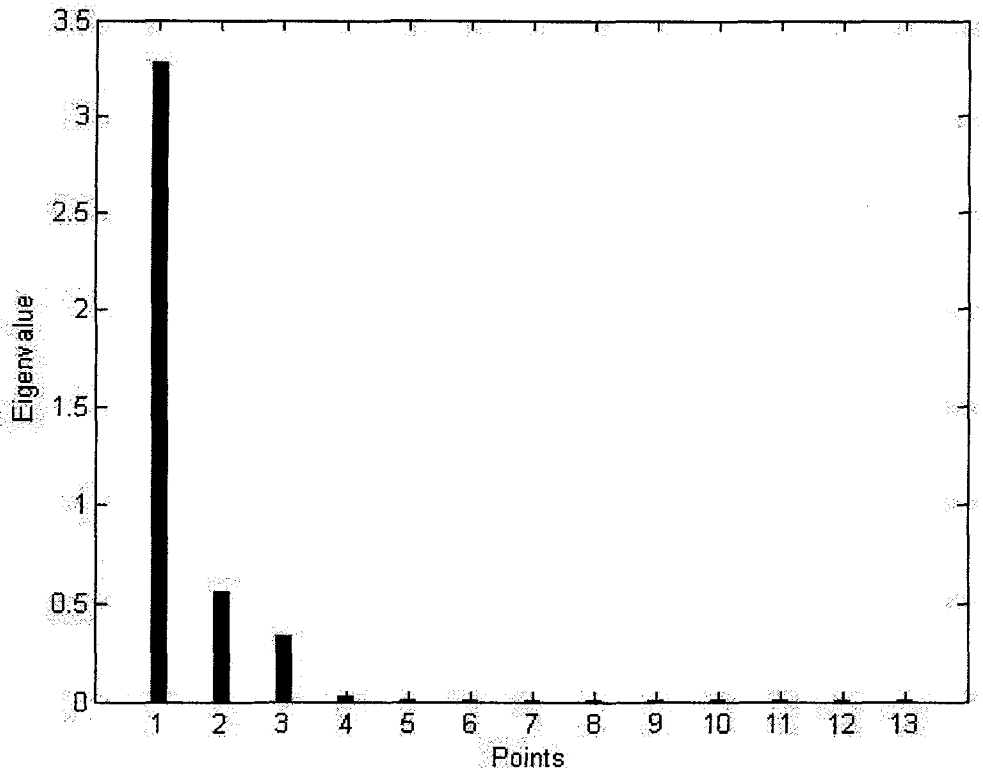


Figure 3.2 Eigenvalues of the sample process variable

The variance explained by each principal component along with the cumulative variance is summarized in Table 3.1. It is apparent that the eigenvalues of the first two principal components can be used as a concise approximation of the original dynamic trend. Therefore, they are used to replace the original responses for use in subsequent pattern recognition.

Since the first two principal components can capture the main feature of a dynamic trend, this can be displayed graphically by plotting the eigenvalues on a two dimensional plane. Figure 3.3 shows such a plot of eigenvalues of the first two principal components of the same variable. A point in the two dimensional plane represents the feature of the variable response trend for one data case.

Table 3-1 Eigenvalues of a process variable

Principal Component Number	Eigenvalue	Variance Described (%)	Cumulative Variance (%)
1	3.2826	77.8257	77.8257
2	0.5623	13.3324	91.1581
3	0.3368	7.9861	99.1442
4	0.0223	0.5283	99.6725
5	0.0056	0.1321	99.8046
6	0.0041	0.0981	99.9027
7	0.0033	0.0793	99.982
8	0.0005	0.0108	99.9929
9	0.0002	0.0046	99.9975
10	0.000042	0.001	99.9984
11	0.000033	0.0008	99.9993
12	0.000029	0.0007	100
13	0	0	100

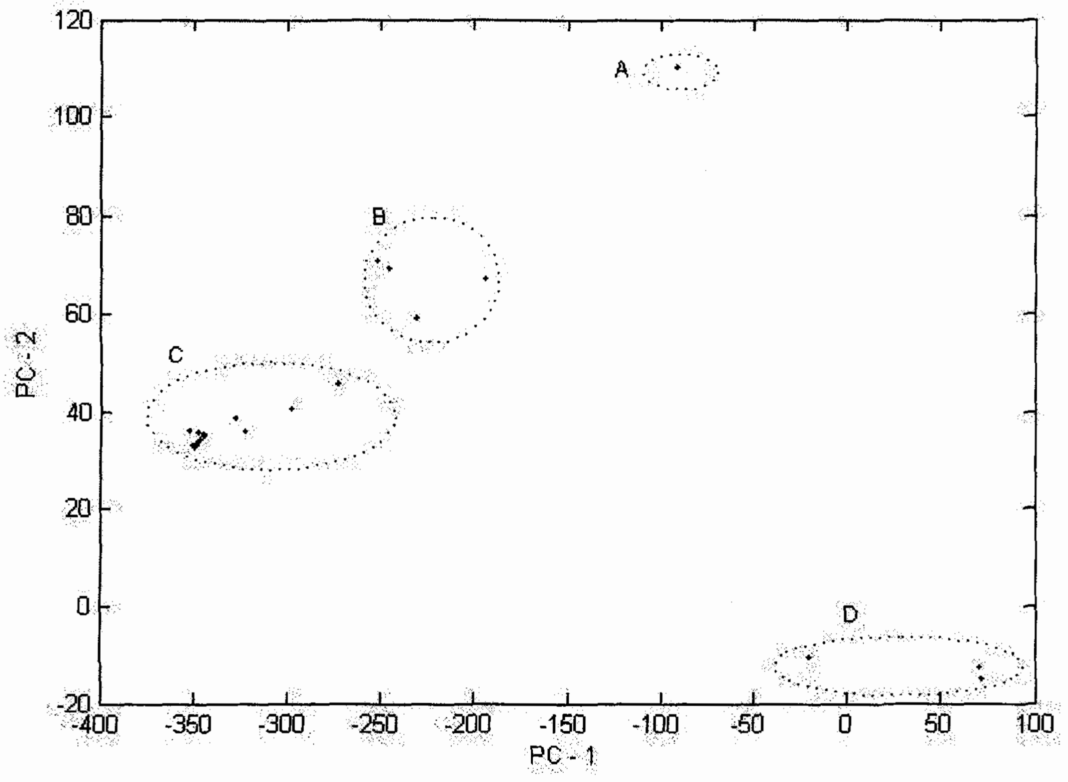


Figure 3.3 PCA two-dimensional plane of a variable

The scatter plot of dynamic trends of a variable on a two dimensional plane, as depicted in Figure 3.3 is referred to as concept formation [1]. Concept formation (or concept extraction from data) transforms a complicated trend to a concept to be used to develop knowledge-based systems. However, this concept formation method proposed by [2] has left the last step to a domain expert to visually group the variables using PCA scatter plot of each process variable. For complex systems with huge number of process variables, the task of grouping based on visual examination of the two-dimensional PCA plane can be very tedious and time consuming.

3.6 Data Clustering – Second Step in the Concept Formation

In this research, concept formation procedure has been converted to an automatic process by assigning the grouping task to the computer. This has been achieved by employing a clustering algorithm to the concept-extraction process to handle grouping. The essential first step in concept formation is to pre-process continuous process variables using PCA model as illustrated in the previous section. The newly introduced second step is to cluster dynamic trends using *k*-Means algorithm. The aim of this modification to the data pre-processing phase is to minimize the intuitive portion of concept formation procedure. This new approach greatly increases the rate at which an expert, or even a novice, can analyze a large and complex dataset. It requires minimal interaction with the domain user to cluster the data using PCA model of the multivariate data set as visualization tool.

3.6.1 Cluster Analysis

The process of grouping physical objects or data points into classes of similar objects is called *clustering* or unsupervised classification. Clustering involves dividing a set of data points into non-overlapping groups, or clusters, of points, where points in the same cluster are "more similar" to one another than to points in other clusters [8]. The term "more similar," when applied to clustered points, usually means closer by some measure of proximity. In clustering, there are no predefined classes and no training data set. The objects are grouped together on the basis of self-similarity. Each cluster is defined as collections of objects whose intraclass similarity is high and interclass similarity is low. When a dataset is clustered, every point is assigned to some cluster, and every cluster can be characterized by a single reference point, usually an average of the points in the cluster. Clustering allows us to replace the original spectral data with an appropriate set of representative values to simplify design and implementation.

As a data mining task, data clustering identifies clusters, or densely populated regions according to some distance measurement, in a large, multidimensional data set. Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. Data clustering identifies the sparse and the crowded regions, and hence discovers the overall distribution pattern of the data set. Clustering analysis has also been studied extensively as a branch of statistics, mainly focused on distance-based clustering [30].

3.6.2 *k*-Means Clustering

The *k*-Means algorithm is by far the most popular clustering tool used in scientific and industrial applications [4]. The name comes from representing each of *k* clusters C_i by the mean (or weighted average) c_i of its points, the so-called centroid. It has the good geometric and statistical sense for numerical attributes.

The *k*-Means algorithm groups the points into *k* clusters such that all the points in each cluster are more similar ("closer") to one another than to those in the other clusters. The number of clusters *k* is chosen by a domain expert or data analyst by examining the PCA plot of a dynamic trend signal on a two dimensional plane and does not require any prior knowledge about process.

k-Means is an iterative algorithm and begins with a set of *k* reference points whose initial values are usually chosen by the user. First, the data points are partitioned into *k* clusters: A data point *x* becomes a member of cluster c_i if r_i, the reference point of cluster c_i, is the reference point closest to *x*. The standard *k*-Means algorithm uses the cluster centroids as reference points in subsequent partitioning. The positions of the reference points and the assignment of the data points to clusters are then adjusted during successive iterations. The error measure *E* is evaluated at each step, and a data point is reassigned to a different cluster only if that reassignment decreases *E*.

To discuss whether a set of points is close enough to be considered in the same cluster, we need a distance measure $D(x; y)$ which tells us how far points *x* and *y* are. The usual axioms for a distance measure *D* are:

1. $D(x; y) = 0$. A point is distance 0 from itself.

2. $D(x; y) = D(y; x)$ Distance is symmetric.

3. $D(x; y) \leq D(x; z) + D(z; y)$. The triangle inequality.

For a *k*-dimensional Euclidean space, the distance between any two points, say $x = [x_1, x_2, \ldots, x_k]$ and $y = [y_1, y_2, \ldots, y_k]$, is given by:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \qquad (3.21)$$

Standard *k*-Means algorithm can be summarized as:

1. Initialize the number of cluster centers selected by the user by randomly selecting them from the training data set

2. Classify the entire training set. For each data point x in the training set, find the nearest cluster centroid c_i and classify x as a member of C_i.

3. For each cluster, recompute its new centroid by finding the mean of the points in each cluster.

4. Repeat steps 2 and 3 until error measure of the centroids don't change.

One of the key design parameters in the standard *k*-Means clustering algorithm is the number of clusters. It assumes that the number of clusters in the data to be clustered is known a priori. However, this may not be a reasonable assumption in many applications. Thus, PCA method used for concept formation serves as a visual tool in the proposed data-mining framework to help find optimum number of clusters for each process variable.

An example of clustering using *k*-Means algorithm is shown in Figure 3.4, Figure 3.5, and Figure 3.6. The diagrams show the results during two iterations in the partitioning of two-dimensional data points of first two PCs of a process variable into two well-separated clusters. Points in cluster 1 are shown in blue, points in cluster 2 are shown in red; data points are denoted by open circles and reference points are denoted by filled circles. Clusters are indicated by dashed lines. It is worth mentioning that the iteration converges quickly to the correct clustering, even for this bad initial choice of the two reference points.

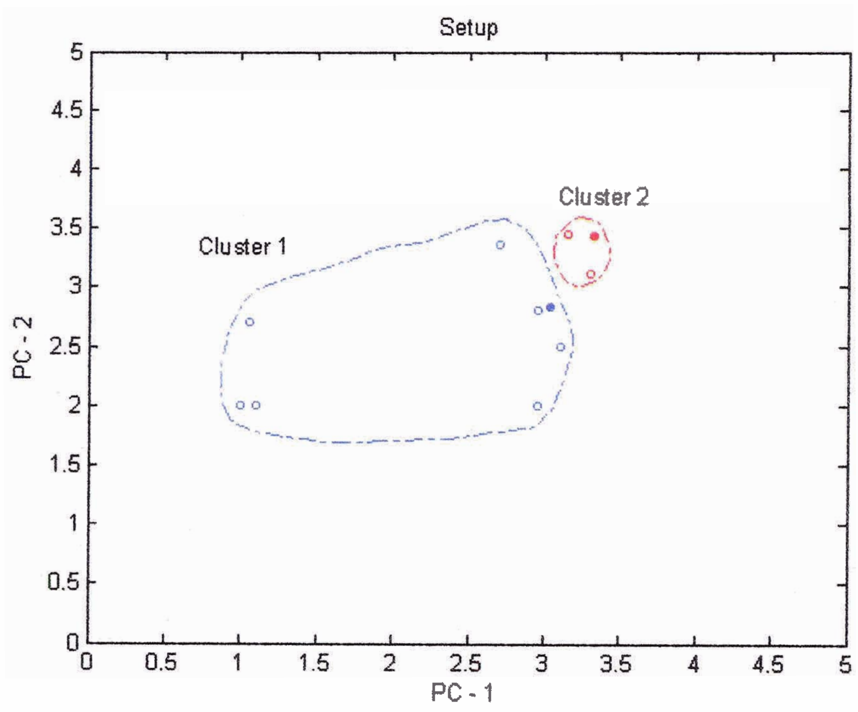


Figure 3.4 Initial setup for k-Means algorithm

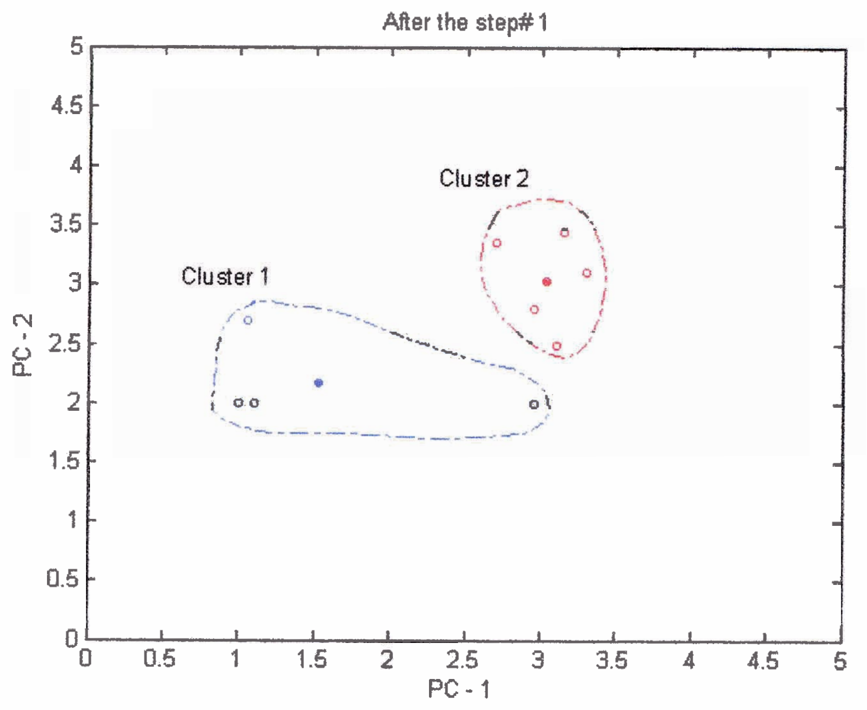


Figure 3.5 Results of the first iteration

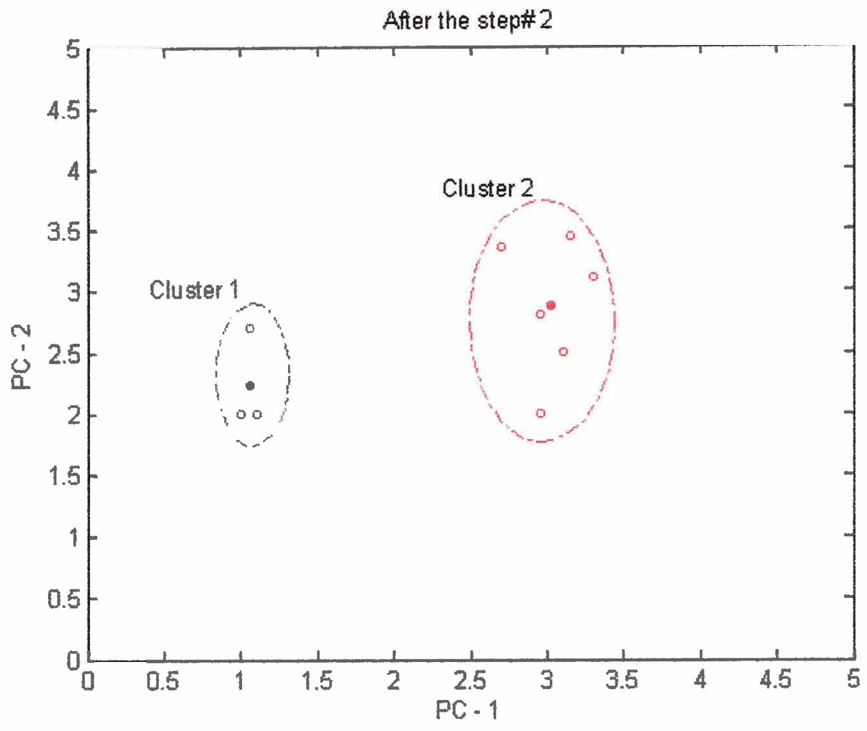


Figure 3.6 Results of the second iteration

During the setup shown in Figure 3.4, reference point 1 (filled blue circle) and reference point 2 (filled red circle) are chosen arbitrarily. All data points (open circles) are then partitioned into two clusters: each data point is assigned to cluster 1 or cluster 2, depending on whether the data point is closer to reference point 1 or 2, respectively. Next, each reference point is moved to the centroid of its cluster. If the reference point closest to the data point belongs to the other cluster, the data point is reassigned to that other cluster, and both cluster centroids are recalculated. The results of the first iteration are shown in Figure 3.5. During the second iteration, the process in Figure 3.5 is performed again for every data point. The partition shown in Figure 3.6 is stable and will not change for any further iteration.

3.7 Conclusions

Concept extraction is not an exact process; even a domain expert can be inconsistent or make mistakes. But by enabling the computer to approximate the domain expert's interpretive skills, concept extraction provides a flexible and rapid way to incorporate the expert's perspective into computer based data analysis and KDD process.

The essential first step in data pre-processing is the PCA analysis of the training data set. For a specific set of data, the value of a variable represents a dynamic trend, consisting of hundreds to thousands sampled points. In the subsequent inductive learning process, it is the shape of the trend that matters. When the trends of all the data sets are considered and processed using PCA, the first two principal components (PCs) can be plotted in a two dimensional plane. Then, the user optionally interacts with the framework, selecting the reference points and number of clusters to guide the clustering process. The framework can also be configured so that no user input is required throughout the concept extraction process enabling the analysis of huge data sets from very complex multivariate processes possible.

CHAPTER 4:
CLASSIFICATION BY DECISION TREE INDUCTION

Historical databases of industrial plants are rich with hidden information that can be used for making intelligent decisions to improve the overall system performance and detect faults. Classification and prediction are two forms of data analysis that can be used to achieve these goals. Classification falls under the category of inductive learning, learning by examples, which attempts to induce a general rule from a set of observed instances. Several approaches to inductive learning have been proposed, and one of the most influential one is C4.5, which was developed by Quinlan [26]. It is a set of computer programs that construct classification models in the form of decision tree.

In this chapter, the algorithm that is used by C4.5 to generate the decision tree will be introduced briefly.

4.1 Decision Tree

A decision tree is a flow-chart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions [4]. Decision trees are powerful tools for classification and prediction. Decision trees can also be expressed as production rules that are easier to understand than a complex tree.

A decision tree is a structure that is either:

- a leaf indicating a class, or

- a decision node, or internal node, that specifies some test to be carried out on a single attribute value, with one branch and subtree for each possible outcome of the test.

After a decision tree has been created based on training data set, it can be used to classify a case from the test set in a process like this: starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the case's outcome for the test at the node is determined based on the case's attribute value and attention shifts to the root of the subtree corresponding to this outcome. Repeat this process until a leaf node is encountered. The class of the case is predicted to be that recorded at the leaf.

A typical decision tree is drawn with the root at the top and the leaves at the bottom. For example, Figure 4.1 shows a decision tree that can be used to determine if a customer would be eligible for a loan. The root node of the tree defines the test "Age of the customer", so each case will be divided first by this test into several groups. The label of a leaf represents the class label. *Yes* or *No* is assigned by the tree to any case that reaches this node. It represents the concept "*loan approval*", that is, it predicts whether or not a customer is likely to get a loan from the bank.

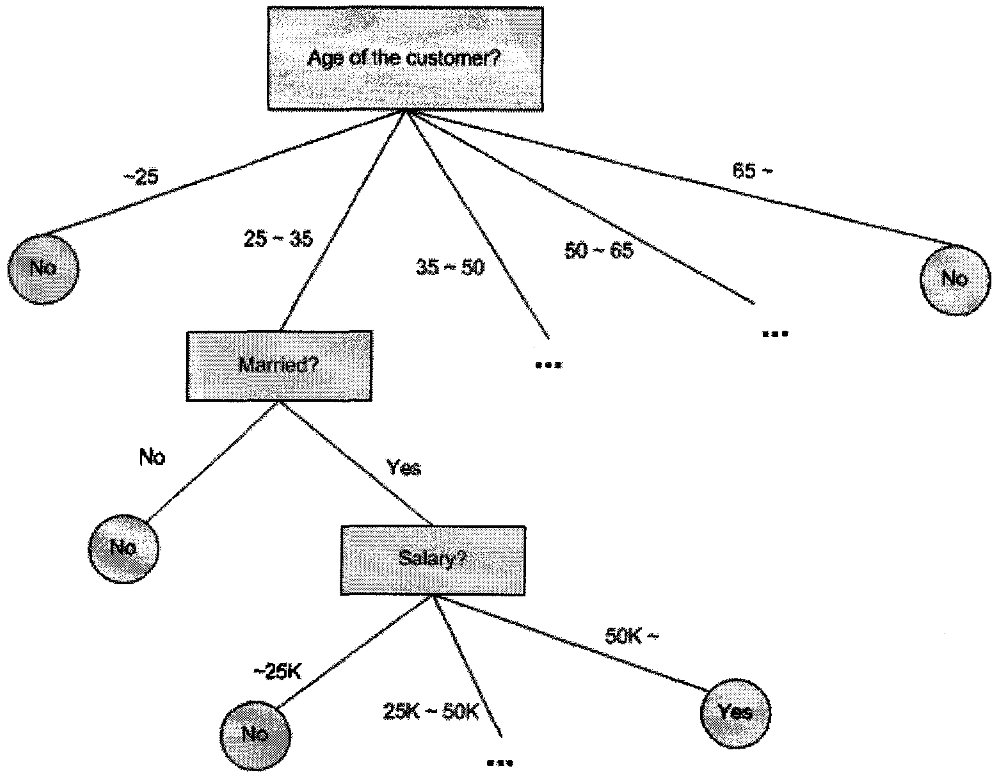


Figure 4.1 A partial decision tree for the concept of "loan approval"

4.2 Decision Tree Induction

The common procedure to construct a decision tree from samples is by using divide and conquers technique. This process aims to discover sizable subsets of the sample that belong to the same class. This technique depends a great deal on the choice of appropriate test, that is, to find the best possible question to ask at each decision node of the tree. The algorithm summarized below is a version of ID3 [Quinlan, 1986], a well-known decision tree induction algorithm that constructs decision trees in a top-down recursive manner:

Let classes be denoted $\{C_1, C_2, \dots, C_3\}$, and T being the set of training cases. There are three possibilities [26]:

- *T* contains one or more cases, all belonging to a single class C_j: The decision tree for *T* is a leaf identifying class C_j.

- *T* contains no cases: The decision tree is again a leaf, but the class to be associated with the leaf must be determined from other information. For example, the leaf might be chosen in accordance with some background knowledge of the domain, such as overall majority class. C4.5 uses the most frequent class at the parent of this node.

- *T* contains examples that belong to more than one class: In this situation, the idea is to divide *T* into subsets of cases so that each subset seems to be heading towards single-class collection of cases. An attribute which has two or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$ is chosen as a test. *T* is partitioned into subsets $T_1, T_2, \ldots, T_n$, where T_i contains all the cases in *T* that have outcome O_i of the chosen test. The decision tree for *T* consists of a decision node identifying the test, and one branch for each possible outcome. The same tree-building process is applied recursively to each subset of training cases, so that the *i*th branch leads to the decision tree constructed from the subset T_i of training cases.

Any test that divides *T* in a way that at least two of the subsets T_i are not empty will eventually result in a partition into single-class subsets. However, the tree building process is not intended merely to find any such partition that will result in single-class subsets, but to build a tree that reveals the structure of the domain and so has predictive power. For that reason, we need a significant number of cases at each leaf; in other words, the partitioning must result in as few blocks as possible so that the final tree is small.

4.3 Attribute Selection

Most decision tree construction methods, including the one described above, are nonbacktracking, greedy algorithms. Once a test has been selected to partition the current set of training cases, the choice is cast in concrete and the consequences of alternative choices are not explored. This is the biggest reason for making the attribute selection process as effective as possible based on maximizing some local measure of progress.

Since exploring all possible decision trees that are consistent with the training set and selecting the simplest is not an option, Quinlan uses a criterion called *gain ratio* to select the best attribute at each partition step in his program C4.5. The information theory that underpins this criterion can be given in one statement: The information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to base 2 of that probability [26]. So, for example, if there are four equally probable messages, the information conveyed by any of them is –log2(1/4) or 2 bits.

Let follow Quinlan's explanation of *gain ratio* criterion: Suppose for an attribute A_i, we have n possible outcomes that partition the set T of training cases into subsets T_1, T_2, ... , T_n. If this test is to be evaluated without exploring subsequent divisions of the T_i's, the only information available for guidance at the moment is the distribution of classes in T and its subsets. Let S be any set of cases, the following discussion will use $freq(C_i, S)$ to denote the number of cases in S that belong to class C_i. The standard notation $|S|$ will be used to denote the number of cases in set S.

Suppose we select one case at random from a set S of cases, and announcing that it belongs to a class Cj. This message has probability

$$\frac{freq(C_j, S)}{|S|} \tag{4.1}$$

and the information it conveys is

$$-\log_2\left(\frac{freq(C_j, S)}{|S|}\right) \text{ bits.} \tag{4.2}$$

To find the expected information from such a message pertaining to class membership, we sum over the classes in proportion to their frequencies in *S*, giving

$$info(S) = -\sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \times \log_2\left(\frac{freq(C_j, S)}{|S|}\right) \text{ bits.} \tag{4.3}$$

When applied to the set of training cases, *info*(*T*) measures the average amount of information needed to identify the class of a case in *T*. The quantity found by Equation 4.3 is also known as the *entropy* of the set *S*.

After *T* has been partitioned in accordance with the *n* outcomes of a test *X*, the expected information requirement can be found as the weighted sum over the subsets, as

$$info_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{T} \times info(T_i). \tag{4.4}$$

So, the quantity

$$gain(X) = info(T) - info_X(T) \tag{4.5}$$

measures the information that is gained by partitioning *T* in accordance with the test *X*. The goal of *gain criterion* is, then, to select a test to maximize this information gain.

4.4 An Illustration

As a concrete illustration of the process, consider the small training set of Table 4.1 in which there are four attributes and two classes [26]. The cases have been grouped on the first attribute *outlook* to simplify the discussion.

Table 4-1 A small training set

Outlook	Temp (°*F*)	Humidity (%)	Windy?	Class
sunny	75	low	true	Play
sunny	80	high	true	Don't Play
sunny	85	moderate	false	Don't Play
sunny	72	high	false	Don't Play
sunny	69	low	false	Play
overcast	72	high	true	Play
overcast	83	moderate	false	Play
overcast	64	low	true	Play
overcast	81	low	false	Play
rain	71	moderate	true	Don't Play
rain	65	low	true	Don't Play
rain	75	moderate	false	Play
rain	68	moderate	false	Play
rain	70	high	false	Play

Since these cases do not all belong to the same class, the divide-and-conquer algorithm attempts to split them into subsets. The successive division of the set of training cases proceeds until all the subsets consist of cases belonging to a single class.

There are two classes, nine cases belonging to *Play* and five to *Don't Play*. The average information needed to identify the class of a case in the set *T* can be calculated using Equation 4.3 as:

info(*T*) = – 9/14 × log2(9/14) – 5/14 × log2(5/14) = 0.940 bits.

Suppose that the test *outlook* with three outcomes, *outlook* = sunny, *outlook* = overcast and *outlook* = rain, is chosen to divide the data set in Table 4.1 into three subsets. The expected information by partitioning the data set *T* with this test is given by

infox(*T*) = 5/14 × (-2/5 × log2(2/5) – 3/5 × log2(3/5))

+ 4/14 × (-4/4 × log2(4/4) – 0/4 × log2(0/4))

+ 5/14 × (-3/5 × log2(3/5) – 2/5 × log2(2/5))

= 0.694 bits.

The information gained by this test is then $0.940 - 0.694 = 0.246$ bits. Now suppose that, instead of dividing *T* on the attribute *outlook*, we had partitioned it on the attribute *windy*. This would have given two subsets, one with three Play and three Don't Play cases, the other with six Play and two Don't Play cases. The similar computation to find the expected information pertaining to this partitioning would be:

infox(*T*) = 6/14 × (-3/6 × log2(3/6) – 3/6 × log2(3/6))

+ 8/14 × (-6/8 × log2(6/8) – 2/8 × log2(2/8))

= 0.892 bits

The information gained would be $0.940 - 0.892 = 0.048$ bits, which is less than the gain resulting from the previous test. The gain criterion would then prefer the test on *outlook* to the latter test on *windy*.

After partitioning the training set of Table 4.1 based on *outlook*, the middle group contains only cases of class *Play* but the first and third subsets still have mixed classes. If the first subset were further divided by a test on *humidity*, with outcomes *humidity* = low and *humidity* in {moderate, high}, and the third subset by a test on *windy*, with outcomes *windy* = true and *windy* = false, each of the subsets would now contain cases from a single class.

For this sample training set of fourteen cases in Table 4.1, it is easy to do partitioning intuitively. The success of decision tree building process depends a great deal on the choice of appropriate tests on appropriate attributes. The final tree must reveal the structure of the domain and have predictive power to successfully classify future cases. The decision tree induction algorithm employed in C4.5 uses gain criterion (or gain ratio criterion which is explained in the next section) to choose a test at each stage of decision tree building process.

The decision tree corresponding to the training set in Table 4.1 is shown in Figure 4.2. C4.5 first splits the training cases into three subsets based on the test on attribute *outlook*. Then, the second and third branches of the tree are further partitioned by the tests on attributes *humidity* and *windy*, respectively. At this stage, each of the subsets contains cases from a single class. Each leaf node is labeled with a class value *Play* or *Don't Play*. The numbers in parentheses following each leaf indicate the number of training cases associated with each leaf.

```
Decision tree:

outlook = overcast: Play (4)
outlook = sunny:
:......humidity = low : Play (2)
:      humidity in {moderate, high} : Don't Play (3)
outlook = rain:
:......windy = true: Don't Play (2)
       windy = false: Play (3)
```

Figure 4.2 Decision tree generated by C4.5

4.5 Gain Ratio Criterion

Although the gain criterion gave quite good results, according to Quinlan, this criterion has a strong bias in favor of tests with many outcomes. We can see this by considering a hypothetical medical diagnosis task in which one of the attributes contains a patient identification. Since every such identification is intended to be unique, partitioning any set of training cases on the values of this attribute will lead to a large number of subsets, each containing just one case. Since all of these one-example subsets contain examples of a single class, *info*x(*T*) = 0, so the information gain from using this attribute to partition the set of training cases is maximal. From the point of view of prediction, however, such a division is quite useless and should be avoided in practice.

Quinlan uses a kind of normalization in which the apparent gain attributable to tests with many outcomes is adjusted. Consider the information content of a message pertaining to a case that indicates not the class to which the case belongs, but the outcome of the test. If we use the analogy with the definition of *info*(*S*), we have

$$\textit{split info}(X) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right) \qquad (4.6)$$

which represents the potential information generated by dividing T into n subsets, whereas the information gain measures the information relevant to classification that arises from the same division. Then,

$$\text{gain ratio}(X) = \text{gain}(X) \,/\, \text{split info}(X) \tag{4.7}$$

expresses the proportion of information generated by the split that is useful, which appears helpful for classification. If the split is near trivial, like the illustration given above with the attribute *patient_id*, split information will be small and this ratio will be unstable. To avoid this, the *gain ratio* criterion selects a test to maximize the ratio above, subject to the constraint that the information gain must be large – at least as great as the average gain over all tests examined. The gain ratio criterion is robust and consistently gives a better choice of test than the gain criterion (Quinlan, 1988).

To illustrate the gain ratio criterion, consider again the training set of Table 4.1. The test on *outlook* produces three subsets containing five, four, and five cases respectively. The split information calculated as

$$\text{split info}(X) = -5/14 \times \log_2(5/14) - 4/14 \times \log_2(4/14) - 5/14 \times \log_2(5/14)$$

or 1.577 bits. For this test, whose gain is 0.246 (as calculated before), the gain ratio is 0.246 / 1.577 = 0.156.

The decision tree induction algorithm adopted in this study uses the gain ratio criterion as it typically gives a constantly better choice of test than the gain criterion (Quinlan, 1988).

4.6 Conclusions

Classification is a data mining technique that can be used to extract models describing important data classes. Since it requires a training data set to model the problem domain, it is also known as supervised learning. In a decision-tree-based classification method, the learned model is represented in the form of decision trees.

A decision-tree-based classification algorithm, C4.5, which has been influential in the machine learning studies, has been introduced in this chapter. The success of any decision tree induction algorithm depends on the measure used for selecting the attribute that will best separate the samples into individual classes. This attribute becomes the "test" or "decision" attribute at the node. The algorithm employed by C4.5 uses an entropy-based measure known as *information gain* as a heuristic to select attributes at each decision node. This attribute selection algorithm has also been explained in detail. Induction of a decision tree by C4.5 has been illustrated by a golf example containing fourteen training cases, four attributes and two classes.

The detailed explanation of the C4.5 algorithm can be found in a book by Quinlan [26].

CHAPTER 5: DATA MINING FRAMEWORK APPLIED TO THREE-TANK BENCHMARK SYSTEM

This chapter presents the validation of the integrated data mining framework by comparing it with traditional multivariate statistical techniques. The proposed framework is illustrated by considering the fault detection problem of the three-tank benchmark system. First, the three-tank benchmark system is introduced briefly along with the various fault types, which are used in this study. Then, a well-known multivariate SPM technique based on principal component analysis is introduced and applied to the benchmark system to detect system faults. Next, the proposed data mining framework is applied to the benchmark system and each step of this process is explained in detail. The performance of the proposed approach is compared with PCA based monitoring technique. Finally, the proposed approach is applied to the benchmark system to demonstrate the fault identification capability of the framework to identify single faults.

5.1 Three-Tank Benchmark System

Many of the faults in chemical processes such as leaks, clogs, valve blockages and sensor faults occur on the level of transport of fluids and raw materials. To study the corresponding diagnostic problems a laboratory desktop plant, composed of tanks interconnected by various hydrodynamic paths, was constructed by the Department of Computer Automation and Control at Jozef Stefan Institute [31]. The plant mimics some of the processes that are common in the transport of fluids in many chemical plants. The schematic of the process is depicted in Figure 5.1 [31]. The three tank system has been adopted recently as a standard benchmark problem for fault detection and diagnostic [39, 40].

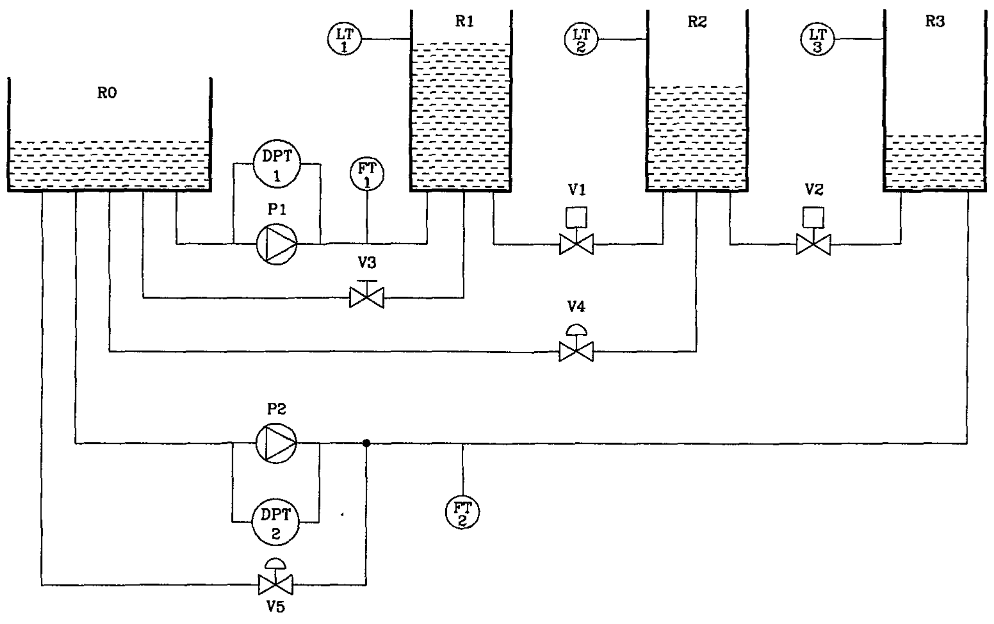


Figure 5.1 Schematic diagram of the benchmark process

The system consists of the three tanks R1, R2 and R3 connected with flow paths, which serve to supply water from the reservoir R0. Two of the paths have built-in pumps, which are pump P1 and P2, driven with DC motors with permanent magnets.

There are two configurations of active flow paths available. In the first one, flow is generated by varying the angular speed of the pump P1. In the second case, pump P2 works at constant speed. Flow is then varied by manipulating the valve V5. There are two servo-valves in the plant, i.e. V4 and V5 driven by DC motors. Valves V1 and V2 are on-off valves while V3 is manual. The purpose of valve V3 is mainly to realize real faults, i.e. leakage of the tank R1. Capacity of the reservoir R0 is much greater than the capacity of the tanks so that its level is practically constant during the operation.

Although the three-tank system is not an equivalent to any of the real industrial processes, it can be studied at different configurations and operating modes. In the study of benchmark system construction, tanks R1 and R3 take on the role of buffers for supplying R2. Contents from R1 and R3 are mixed in R2 and then fed back to the reservoir R0. The level in R2, and hence the flow from R2 to R0, is controlled by the valve V4. The level in the tank R1 is controlled by manipulating the speed of the pump P1 while level in R3 is controlled by manipulating the command signal of the valve V5. Proper ratios of flows from R1 into R2 and from R3 into R2 are achieved by adjusting the difference of the reference values of levels in the tanks.

The benchmark consists of the Simulink file with nonlinear simulation model of the plant. The Simulink module of the system is shown in Figure 5.2 [31]. The model can simulate 20 different faults in sensors and other components, which can be either *real* or *virtual*. Some of the real faults are:

a) leak from the tank R1 (by opening the manual valve V3)

b) clog in branch with V1

c) clog in branch with V2

d) increased friction in the pumps

e) offsets in sensors

Faults a, b and c can be programmed using the Simulink model of the system. Faults d and e might occur during long-term runs. Virtual faults include:

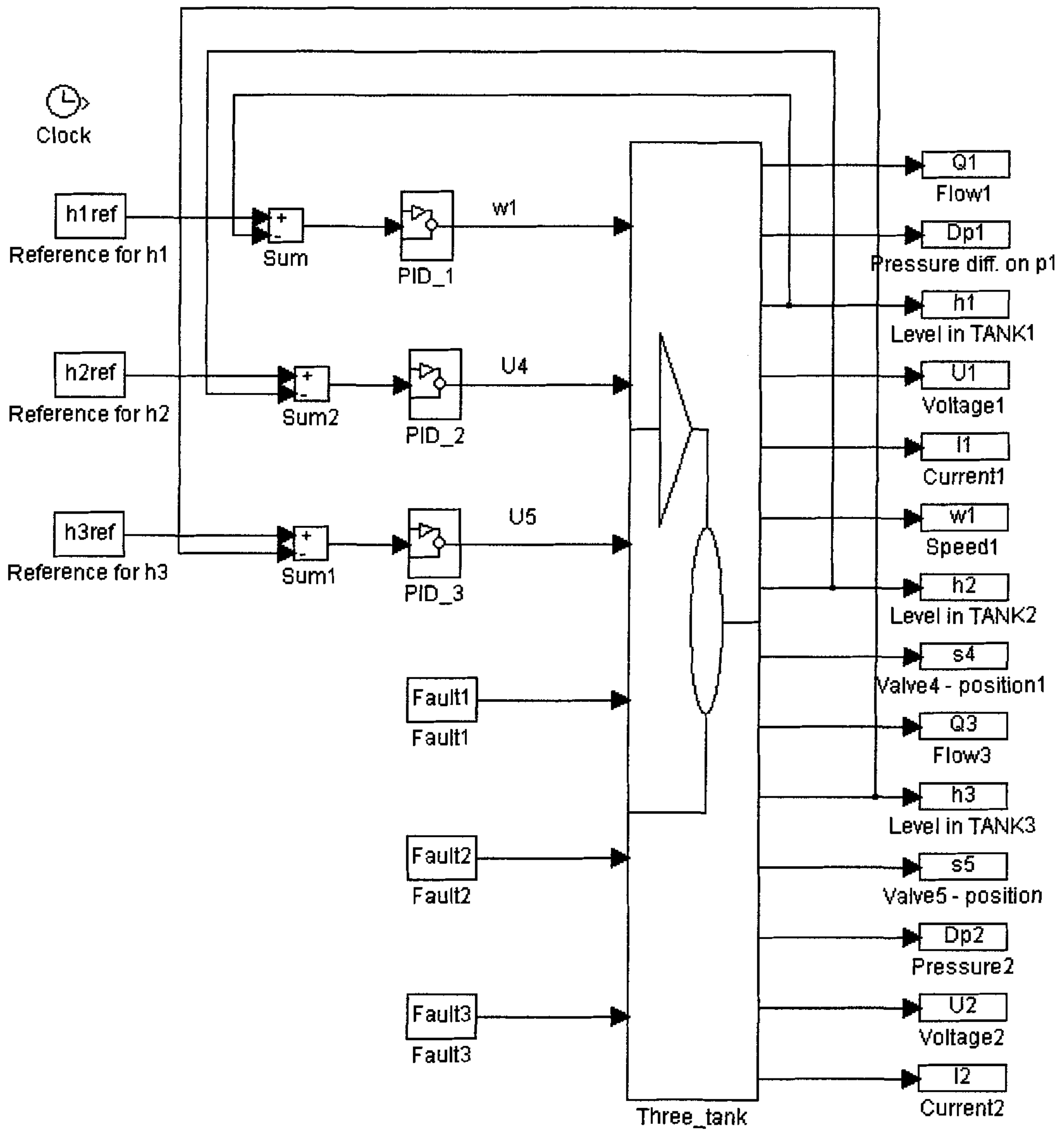


Figure 5.2 Simulink schema of the system

- sensor faults, e.g. biases, change in gain
- actuator faults, e.g. blockages in valves

They can be realized by contaminating the realistic measured values.

A demo file simulating three system faults (bias in sensor of h3 started at 200 seconds, leak in tank R1 started at 500 seconds, clog in branch with P2 started at 900 seconds and each fault lasted for 100 seconds) is given in Figure 5.3.

```
% Data preparation for simulation

% Definition of the initial conditions for the integrators
h1_0    = 0.;                % level in R1
h2_0    = 0.;                % level in R2
h3_0    = 0.;                % level in R3

% Definition of the reference trajectories

h1ref   = [0 35; 1200 35];
h2ref   = [0 30; 1200 30];
h3ref   = [0 35; 1200 35];

% Setting of the on-off valves

sDV1 = 10;          % valve V1 open
sDV2 = 10;          % valve V2 open

% Definition of the fault channels

biash3  = [0  0;  199.9  0;  200   0.8;  299.9  0.8;  300   0;  1200  0];
leakR1  = [0  0;  499.9  0;  500   0.2;  599.9  0.2;  600   0;  1200  0];
clogP2  = [0  0;  899.9  0;  900   0.3;  999.9  0.3;  1000  0;  1200  0];

% Start and end of simulation
Tstart  = 0;
Tend    = 1200;

% Set the controller parameters
Kp1     = 2;
Ti1     = 100;
Kp2     = 5;
Ti2     = 100;
Kp3     = 5;
Ti3     = 100;

% Run the data initialization procedure

ini3tank
```

Figure 5.3 A demo simulation file realizing 3 different faults.

The demo simulation file given in Figure 5.3 prepares the input data (define initial conditions, fault signals and reference trajectories). It then runs the initialization program ini3tank.m which sets all the constants of the model and adds noise to all measurements. Then in Simulink, simulation program of the system operating in closed loop is run. Detailed description of the simulation environment developed in Simulink to reproduce the system behavior under faulty or fault free situations along with the derivation of the nonlinear mathematical model of the system can be found in [31].

Measurements of the 13 process variables in Table 5.1 were used to generate historical process data. The system can be operated either in open loop or closed loop. In closed loop, levels in the tanks R1, R2 and R3 are read via sensors and fed back to the controllers along with the reference trajectories for the tank levels defined at the beginning of the simulation. In this case study, the closed-loop model of the system is used under influence of noise and faults. Zero-mean Gaussian noise with known standard deviations is added to the measured process variables during both normal and faulty operations.

Table 5-1 Available process measurements

Process Variable	Unit	Description
h_1	cm	level in tank R1
h_2	cm	level in tank R2
h_3	cm	level in tank R3
Δp_1	cm H_20	pressure difference on the pump 1
Δp_2	cm H_20	pressure difference on the pump 2
Q_1	cm^3/s	flow through the pump P1
Q3	cm^3/s	flow through the branch with pump P2
ω_1	-	"speed" of rotation of pump P1
s_5	-	position of the stem of the continuous valve V5
U_1	V	voltage on the DC motor in pump P1
I_1	-	current to the DC motor in P1
U_2	V	voltage on the DC motor in pump P2
I_2	-	current to the DC motor in P2

There are three feedback loops in the system. The first loop controls the level of tank R1 by adjusting the speed ω_1 of pump P1, the second loop controls the level of tank R2 by adjusting the position of valve V4, and the third loop controls the level of tank R3 by adjusting the valve V5. The overall model consists of three control inputs and thirteen outputs along with the optional fault signals. Table 5.2 lists the 20 faults applied to the system during the process simulation. The intensity of each fault can be defined in the range of 0 to 1.

The Simulink® model, which was developed by Jozef Stefan Institute, was used in Matlab 7® to simulate the different operation conditions of the system. Reference trajectories (tank levels) were kept constant throughout the study.

Table 5-2 List of simulated faults

Fault No	Fault Description
1	bias in sensor of h1
2	bias in sensor of h2
3	bias in sensor of h3
4	leak in R1
5	clog in branch with V1
6	clog in branch with V2
7	clog in branch with V4
8	clog in branch with P1
9	clog in branch with P2
10	friction in P1
11	clog in branch with V5
12	bias in sensor of Q1
13	bias in sensor of Dp1
14	bias in sensor of Dp2
15	bias in sensor of I1
16	bias in sensor of U1
17	bias in sensor of Q3
18	bias in sensor of I2
19	bias in sensor of U2
20	bias in sensor of w1

5.2 Traditional Approach: PCA Based Fault Detection

Multivariate statistical process monitoring (MSPM) techniques such as principal component analysis (PCA) have been successfully employed in many industrial applications for abnormal situation detection and fault diagnosis [11]. In contrast to the model-based approaches where a priori knowledge (either quantitative or qualitative) about the process is needed, in PCA based method, only the availability of historical process data is needed. The primary objectives of PCA are data summarization, classification of variables, outlier detection and 'fingerprinting' for fault identification [12].

Traditional PCA based process monitoring for fault detection requires constructing multivariate control charts such as Q statistic. The Q statistic, also called as squared prediction error (SPE), describes how far a measurement lies from subspace defined by the PCA model. Faults that result in a change in the cross-correlation between process variables can be detected by monitoring the Q statistic.

Multivariate process monitoring using Q statistic involves following steps: First, an appropriate reference set that defines the normal (routine) operating conditions for a particular process is chosen. In other words, a PCA model must be built based on data collected from various periods of plant operation when performance was good. By projecting new observations of process variables onto the plane defined by the PCA loading vectors, the score and the residuals can be obtained, and the multivariate process control chart based on Q-statistic can in turn be plotted. The Q statistic is calculated and compared to confidence limits at each sampling time in order to determine if a measurement has deviated from the normal operating region.

The computation of Q statistic along with the confidence limits was explained in detail in Chapter 3.

5.2.1 PCA Model of the Three-Tank System

Data from the normal operating conditions were created for the three-tank system to provide the nominal (reference) data set. Any periods containing variations arising from special events or faults that one would like to detect in the future were omitted at this stage.

To build the PCA model of the three-tank benchmark system, simulation was performed under normal operating conditions collecting 200 measurements of all 13 variables. The data matrix X of size (200×13) was used to calculate PCA loading vectors. The principal component loading vectors are the eigenvectors of the covariance matrix of X. The corresponding eigenvalues give the variance of the principal components.

Let S be the covariance matrix of X. The characteristic roots can be obtained from the solution of the following equation, called the characteristic equation:

$$| S - \lambda I | = 0 \tag{5.1}$$

For the three-tank benchmark system, this equation produces a 13th degree polynomial in λ from which the values of λ_1, λ_2, ... λ_{13} are obtained. Then, the characteristic vectors of the covariance matrix can be obtained by solving the following two equations:

$$| S - \lambda_i I | \, t_i = 0 \tag{5.2}$$

and

$$u_i = \frac{t_i}{\sqrt{t_i' t_i}} \qquad \text{for } i = 1,2,3,....13 \tag{5.3}$$

The characteristic vectors or eigenvectors make up the following matrix:

$$U = [u_1 \ u_2 \ \dots \ u_{13}] = \begin{bmatrix} -0.0066 & 0.0082 & & 0.0005 \\ -0.0004 & 0.0019 & & -0.0006 \\ 0.0009 & 0.0012 & & 0.8931 \\ 0.1234 & -0.9803 & & 0.0000 \\ 0.9922 & 0.1205 & & 0.0000 \\ 0.0110 & -0.0126 & & 0.0013 \\ 0.0000 & -0.0006 & \dots\dots\dots & -0.0023 \\ 0.0007 & -0.0097 & & 0.0009 \\ -0.0016 & 0.0089 & & 0.0002 \\ -0.0017 & -0.0024 & & 0.4493 \\ -0.0106 & -0.1553 & & -0.0001 \\ -0.0005 & 0.0013 & & 0.0033 \\ 00002 & 0.0003 & & 0.0201 \end{bmatrix}$$

To calculate the principal components of the data set X, the following transformation is used:

$$z = U'[x - \bar{x}] \qquad (5.4)$$

Here x and $\bar{x}$ are 13×1 vectors of observations on the original variables and their means. Each of 200 observations (or measurements) is transformed to build the PCA model. The individual transformed observations are called *z*-scores.

The number of principal components to retain in the PCA model was assessed based on the amount of the explained variability. The variance explained by first five PCs along with the cumulative variance is summarized in Table 5.3.

Table 5-3 Variance explained by first five principal components

Principal Component Number	Variance Described (%)	Cumulative Variance (%)
1	77.83	77.83
2	13.33	91.16
3	7.99	99.15
4	0.53	99.68
5	0.13	99.81

The first two PCs explain 91% of the variance in the data providing an adequate description of the total variance in the system. Only the first two PCs were retained during the calculation of PCA model. The scores which describe where the original data points project in the PCA subspace were calculated using the first two eigenvectors.

One of the observations on the original variables of the three-tank system is:

$$x = [\ 34.98\ \ 8.13\ \ 25.01\ \ 11.12\ \ 46.98\ \ 3.73\ \ 0.01\ \ 0\ \ 10.57\ \ 2.70\ \ 35.72\ \ 4.66\ \ 0.16\ ]'$$

and the means of original variables are:

$$\bar{x} = [\ 34.99\ \ 8.11\ \ 25.00\ \ 11.68\ \ 47.00\ \ 3.80\ \ 0\ \ 0.04\ \ 10.57\ \ 2.72\ \ 35.64\ \ 4.67\ \ 0.16\ ]'$$

Substituting in (5.4) produces:

$$z = [u_1 u_2]'[x - \bar{x}] = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} -3.06 \\ 0.70 \end{bmatrix}$$

The first two eigenvectors, u_1 and u_2, were used to calculate the principal components of each observations in the nominal data set. After principal components (or z-scores) of each individual observations have been calculated, principal component 1 (z_1) can be plotted against principal component 2 (z_2) on a two dimensional plane.

Figure 5.4 shows the projection of sample process data taken from normal operating conditions onto the two dimensional subspace defined by principal component 1(PC 1) and principal component 2 (PC 2). It also depicts the geometric interpretation of a score vector for a data point projected in the subspace. The score vectors simply define the projection of the sample points onto each eigenvector (loading vector) and thus describe the location of each observation in the PCA subspace.

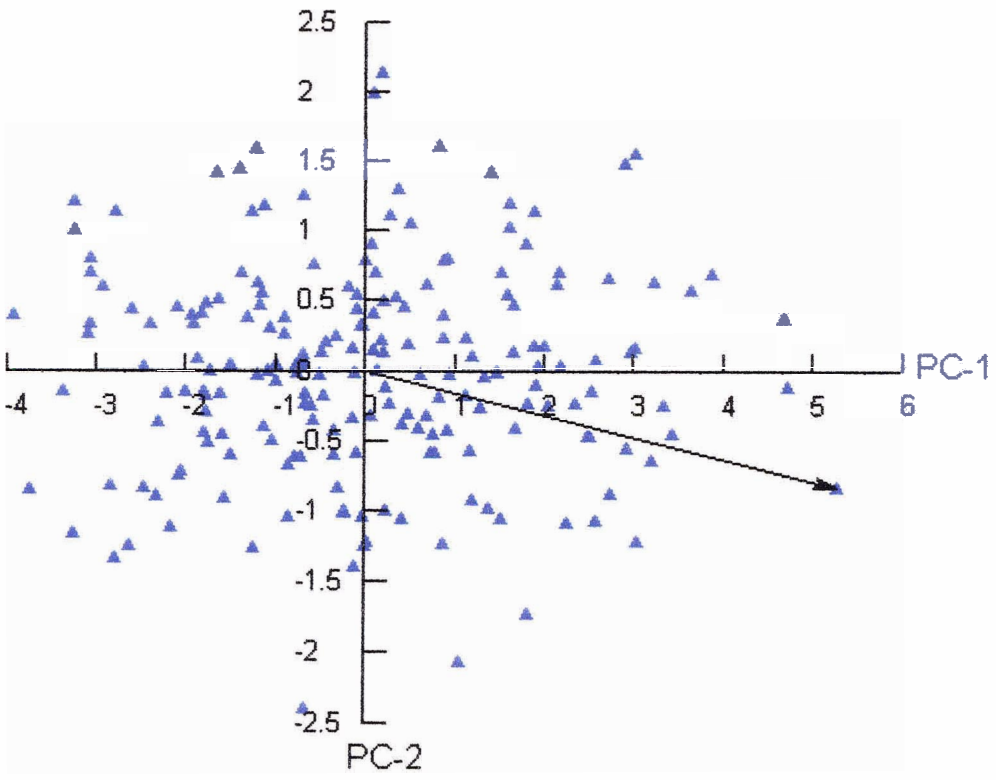


Figure 5.4 Geometric interpretation of a score vector for a data point projected on the two dimensional subspace defined by PC1 and PC2

Figure 5.5 shows the projection of process data taken from two different modes of operation (triangles and circles). Triangles represent the data taken from normal process operation and circles from a faulty operation. This score plot shows the projection of all the process variables on the first two principal components. Score plots also show how each data sample relates to one another. Samples which are in close proximity to one another have similar characteristics or come from similar modes of operation in the process.

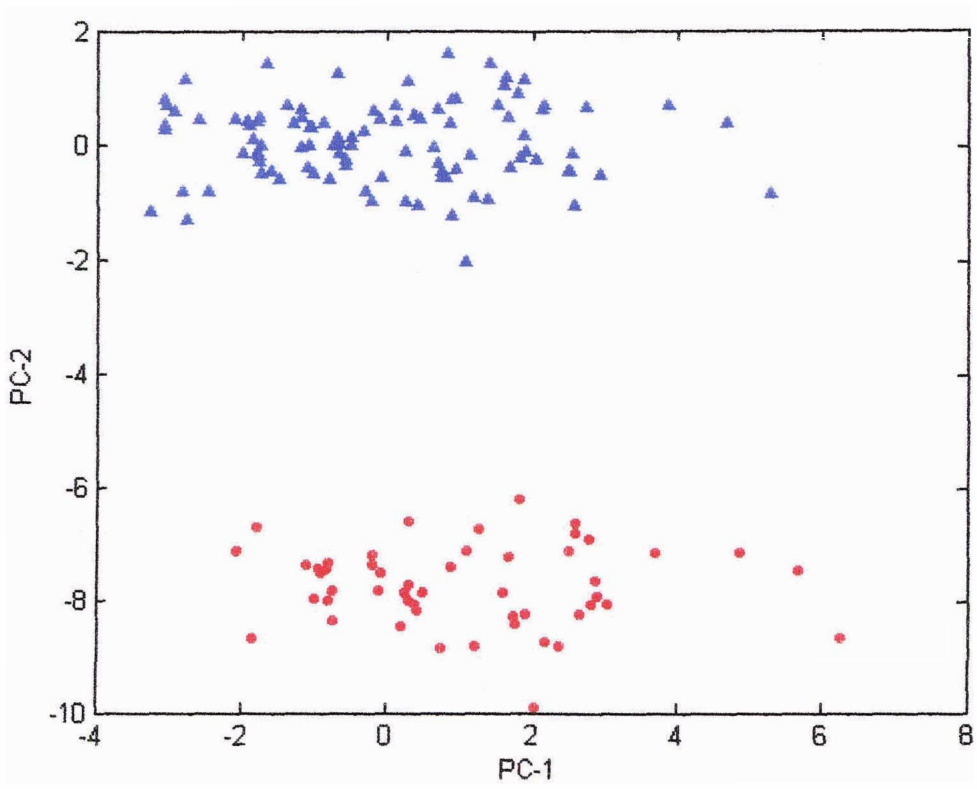


Figure 5.5 Scores plot for principal component 1 versus principal component 2, calculated from nominal data (triangles) and faulty process operation

5.2.2 Fault Detection based on Q statistic

Once a PCA model has been built based on historical data, process monitoring can be achieved by comparing the factors against this nominal model [22].

To compare a new data set containing *m* measurements of *n* variables to a PCA model, the Q-statistics is calculated for each sample, resulting in an $m \times 1$ vector of Q values. This vector, V_Q, in combination with the confidence limit, can be plotted to determine whether the PCA model is an adequate description of the data set. If a deviation from the system model is detected within the predefined limits, this can be an indication that the new observation is the result of previously unidentified event, which represents a faulty operation mode.

5.2.2.1 Preparing the Data Set

Once the PCA model of the three-tank system was created, each of the faulty operations was simulated to generate a test data set. A disturbance or fault was introduced using the fault channels defined in the Simulink model. The system was run under normal operation mode initially and a fault was introduced into the system for 100 seconds. Starting at the time the fault was introduced, dynamic responses of all thirteen variables were recorded for 200 seconds. This same method repeated for all faults listed in Table 5.2. Faults occurred one at a time (no simultaneous faults) in each data case and for the same length of time. The resulting data set is a matrix of size $(200 \times 13 \times 20)$ where the third dimension represents the number of faulty operations.

5.2.2.2 Calculating the Q-statistic

Once the data set containing data from all faulty operation modes was created, the Q statistic values for each operation mode were calculated by projecting each data case onto the PCA model and calculating residuals at each sampling point.

As an illustration, let us assume that the benchmark system is simulated by introducing one of the faults listed in Table 5.2. The dynamic responses of all thirteen variables are recorded every second for 200 seconds after the fault is introduced into the system. For one faulty operation, the data set is a matrix of 200×13. Let the vector x, of size (13×1), represent one of the 200 observations recorded during the simulation.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ x_{13} \end{bmatrix} = \begin{bmatrix} h1 \\ h2 \\ h3 \\ Q1 \\ \Delta p1 \\ U1 \\ I1 \\ \omega 1 \\ Q3 \\ s5 \\ \Delta p2 \\ U2 \\ I2 \end{bmatrix} = \begin{bmatrix} 35.0207 \\ 7.8178 \\ 34.6262 \\ 10.6430 \\ 44.1978 \\ 3.7285 \\ 0.0033 \\ 0.0241 \\ 9.2771 \\ 0 \\ 31.9457 \\ 4.6754 \\ 0.1621 \end{bmatrix}$$

where each row represents the value taken by one of the system variables.

Since the original variables are in different units, they have to be scaled in a meaningful way. In this study, characteristic vectors are scaled in a way that scores will have unit variances. This scaling technique is quite popular for data analysis and quality control applications [7].

To scale the U-vectors (characteristic vectors or eigenvectors), the following transformation have been used:

$$W = UL^{-1/2} \tag{5.5}$$

where L is a diagonal matrix and has characteristic roots (or eigenvalues) of the covariance matrix S as diagonal elements. The L matrix has been calculated during the PCA model building as:

$$L = \begin{bmatrix} 3.2826 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5623 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3368 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0223 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0056 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0041 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0033 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0002 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Principal components obtained by the transformation:

$$y_i = w_i'[x - \bar{x}] \tag{5.6}$$

will produce PCs that are still uncorrelated but now have variances equal to unity. Values of this quantity are called *y*-scores. The relation between *y*- and *z*-scores is:

$$y_i = \frac{z_i}{\sqrt{l_i}} \tag{5.7}$$

where l_i is the *i*th characteristic root (or eigenvalue) of the covariance matrix. Since we have decided to retain the first two PCs from the PCA model, first two columns of W will be used to calculate the *y*-scores.

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}'[x - \bar{x}]$$

$$= \begin{bmatrix} -0.0037 & 0.0109 \\ -0.0002 & 0.0025 \\ 0.0005 & 0.0016 \\ 0.0681 & -1.3072 \\ 0.5476 & 0.1606 \\ 0.0061 & -0.0167 \\ 0.0000 & -0.0008 \\ 0.0004 & -0.0129 \\ -0.0009 & 0.0119 \\ -0.0009 & -0.0032 \\ -0.0058 & -0.2071 \\ -0.0003 & 0.0018 \\ 0.0001 & 0.0004 \end{bmatrix}' \begin{bmatrix} 35.0207 & 34.9999 \\ 7.8178 & 8.1158 \\ 34.6262 & 25.0007 \\ 10.6430 & 11.6885 \\ 44.1978 & 47.0035 \\ 3.7285 & 3.8047 \\ 0.0033 & - & 0.0048 \\ 0.0241 & 0.0416 \\ 9.2771 & 10.5742 \\ 0 & 2.7247 \\ 31.9457 & 35.6425 \\ 4.6754 & 4.6671 \\ 0.1621 & 0.1613 \end{bmatrix} = \begin{bmatrix} -1.5783 \\ 1.6917 \end{bmatrix} \tag{5.8}$$

The equation (5.4) may be inverted so that the original variables may be stated as a function of the principal components:

$$x = \bar{x} + Uz \tag{5.9}$$

because U is orthonormal and hence $U^{-1} = U'$. This means that, given the z-scores, the values of the original variables may be uniquely determined [7]. However, x will be determined exactly only if all the PCs are used. If $k < p$ PCs are used, only an estimate $\hat{x}$ of x will be produced. For the three-tank benchmark system, these values are $k = 2$, which is the number of PCs retained in the PCA model, and $p = 13$, total number of PCs. The estimate of x is

$$\hat{x} = \bar{x} + Uz \tag{5.10}$$

Using the equations (5.5) and (5.7), this equation can be rewritten as

$$\hat{x} = \bar{x} + Vy \tag{5.11}$$

where V -vectors are defined as

$$V = UL^{1/2} \tag{5.12}$$

The predicted test values, given the PCs in (5.8), are calculated using the first two columns of V as:

$$\hat{x} = \bar{x} + Vy = \begin{bmatrix} 34.9999 \\ 8.1158 \\ 25.0007 \\ 11.6885 \\ 47.0035 \\ 3.8047 \\ 0.0048 \\ 0.0416 \\ 10.5742 \\ 2.7247 \\ 35.6425 \\ 4.6671 \\ 0.1613 \end{bmatrix} + \begin{bmatrix} 35.0207 & 34.9999 \\ 7.8178 & 8.1158 \\ 34.6262 & 25.0007 \\ 10.6430 & 11.6885 \\ 44.1978 & 47.0035 \\ 3.7285 & 3.8047 \\ 0.0033 & - \; 0.0048 \\ 0.0241 & 0.0416 \\ 9.2771 & 10.5742 \\ 0 & 2.7247 \\ 31.9457 & 35.6425 \\ 4.6754 & 4.6671 \\ 0.1621 & 0.1613 \end{bmatrix} \begin{bmatrix} -1.5783 \\ 1.6917 \end{bmatrix} = \begin{bmatrix} 35.0293 \\ 8.1193 \\ 24.9997 \\ 0.0921 \\ 44.3191 \\ 3.7572 \\ 0.0040 \\ 0.0274 \\ 10.5901 \\ 2.7265 \\ 35.4757 \\ 4.6704 \\ 0.1613 \end{bmatrix}$$

The residuals are

$$x - \hat{x} = \begin{bmatrix} -0.0085 \\ -0.3015 \\ 9.6264 \\ 0.5509 \\ -0.1212 \\ -0.0287 \\ -0.0007 \\ -0.0033 \\ -1.3130 \\ -2.7265 \\ -3.5300 \\ 0.0051 \\ 0.0008 \end{bmatrix}$$

and their sum of squares (or Q statistic) is

$$Q = (x - \hat{x})'(x - \hat{x}) = 114.6969 \qquad (5.13)$$

Q statistic was calculated for each sampling point generating a vector, **V$_Q$**, of size (200×1). The same calculation was repeated for each type of faulty operation resulting in 20 **V$_Q$** vectors of size (200×1). Then, 20 **V$_Q$** vectors were compared to the confidence limits for the Q statistic to determine whether the corresponding data set points to a faulty operation.

5.2.3 Monitoring of the Three-Tank System for Fault Detection

The Q statistic describes how far a measurement lies from the subspace defined by the PCA model. If the number of times that the corresponding **V$_Q$** vector exceeds the 99% confidence limits for the Q statistic, **Q$_{99}$**, is more than 2, the current data set is considered to come from a faulty operation. (The current data set contains 200 measurements; therefore, the 99% confidence limits should be exceeded more than twice.)

In the three-tank benchmark system, the limit for Q statistic can be calculated using the fact that the first two PCs were retained. The last $p - k = 11$ roots can be substituted in equations (3.17) and (3.18). From these, $\theta_1 = 0.3729$, $\theta_2 = 0.1140$, $\theta_3 = 0.0382$, and $h_0 = 0.2689$. Letting $\alpha = 0.01$ and $c_\alpha = 2.57$, which corresponds to a 99% confidence limit, the limit for Q, using the equation (3.16), is calculated as:

$$Q_{.01} = (0.3729)\left[\frac{(2.57)\sqrt{(2)(0.1140)(0.2689)^2}}{0.3729} + \frac{(0.1140)(0.2689)(-0.7311)}{(0.3729)^2} + 1\right]^{\frac{1}{0.2689}}$$

$$= 2.8252$$

Values of Q higher than this are an indication that a data vector cannot be adequately represented by a two-component model. The Q statistic value for the sample observation calculated in (5.13), 114.6969, is significant according to this limit, which means a faulty operation is detected.

This monitoring technique was evaluated for the 20 possible types of faulty operations described in Table 5.2. A process fault with intensity of 0.2 was introduced into the system at 0 second and lasted for 100 seconds. The process was monitored based on Q statistic with 99% confidence limits for 200 seconds. The results from the simulation of each 20 types of faults are given in Figure 5.6 to Figure 5.9. The dotted red line on each figure shows the 99% confidence limit that has to be exceeded.

This simulation study shows how PCA model of the three-tank system does in detecting single faults occurring under normal operating conditions. The results show that PCA based model detects 14 faults out of 20 at this intensity level corresponding to 70% success rate. It cannot detect the following faults given this confidence level: leak in R1, clog in branch with P2, clog in branch with P1, increased friction in P1, bias in sensor of I1, and bias in sensor of I2 (see Figure 5.6 (2,3), Figure 5.7 (7,8), and Figure 5.8 (14,18)).

The same simulation was also run for fault intensity 0.5 and 1. The Q statistic plots of these simulations are given in Appendix 1 and 2. When the intensity of the faults increased to 0.5, the PCA based model can detect 15 faults out of 20 faults increasing the success rate to 75% by detecting the fault “leak in R1”, which was missed when the fault intensity was at 0.2. When the intensity of the fault is increased to 1, the performance of the traditional approach gets a little bit better reaching to 80% success rate by detecting 16 out of 20 faults. The new fault “clog in branch with P1” gets detected in addition to those already detected.

The traditional PCA based monitoring and fault detection worked quite well for detection certain types of faults when fault intensity was high. However, the performance of the PCA model degrades very quickly as we lower the intensity of the faults. The poor performance can be largely attributed to differences in the variance from one fault to another; so that data sets with smaller variances will appear to fall within the confidence region of the PCA model, regardless of differences in the covariance structure.

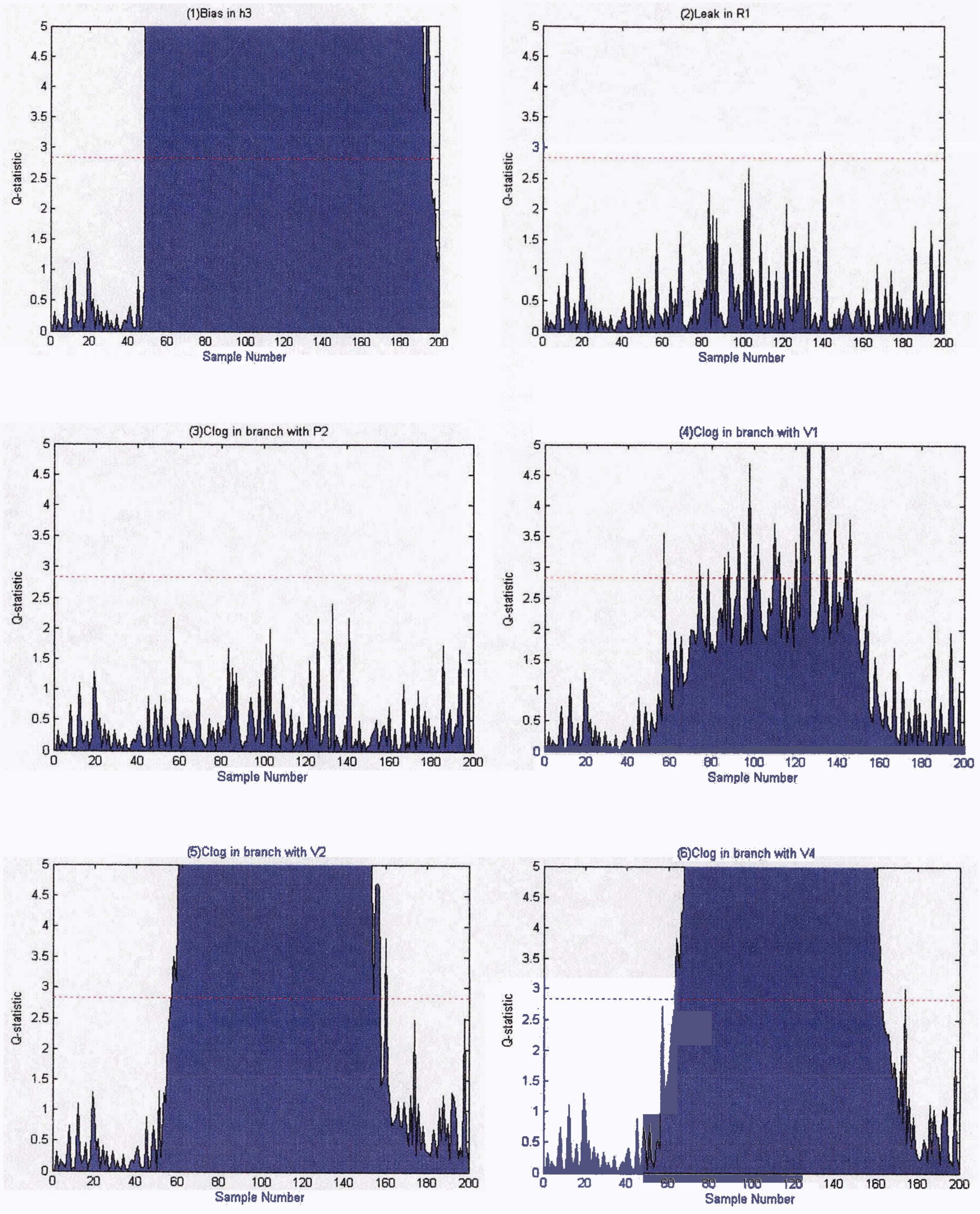


Figure 5.6 Q-statistic plots with 99% confidence limit for fault intensity of 0.2

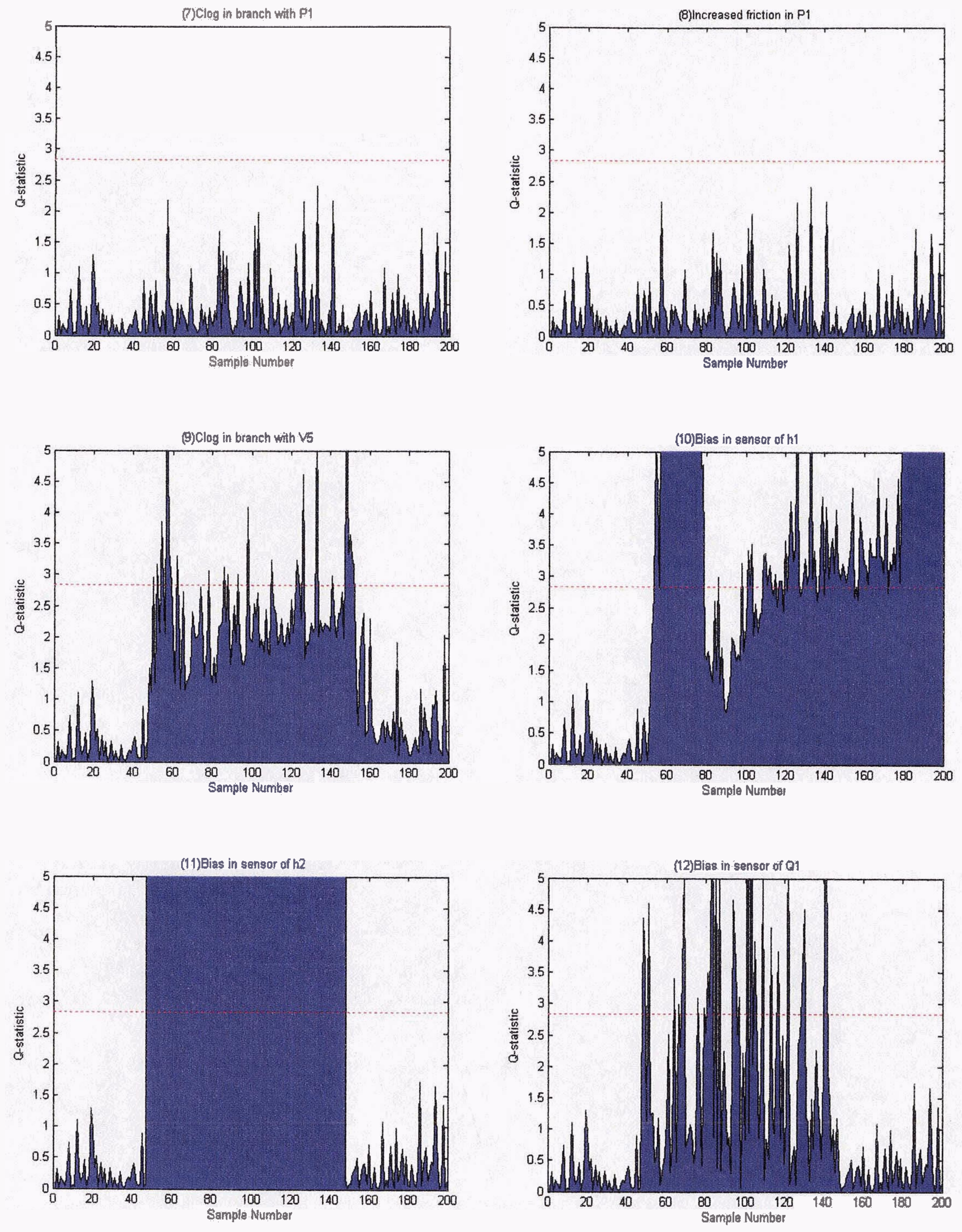


Figure 5.7 Q-statistic plots with 99% confidence limit for fault intensity of 0.2

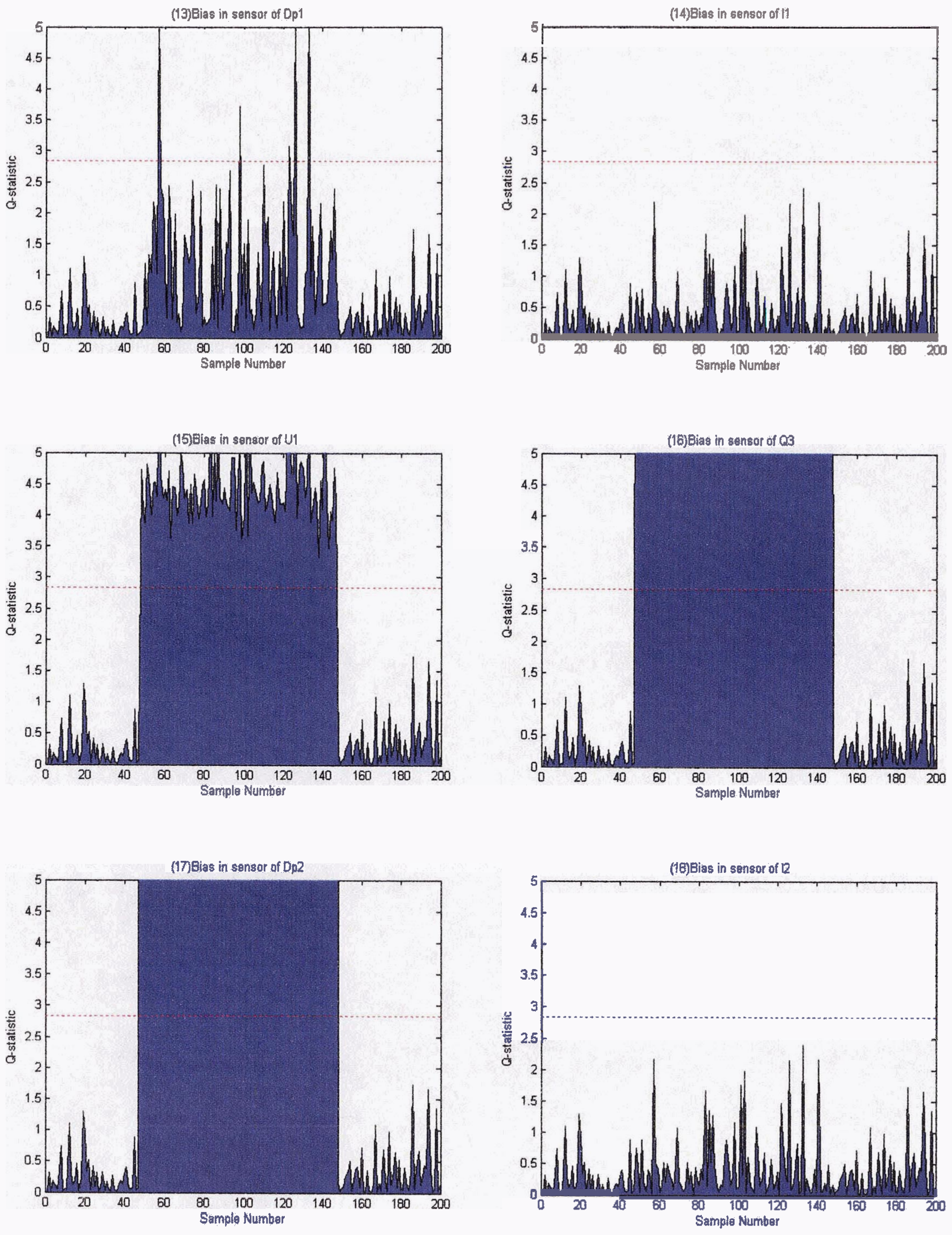
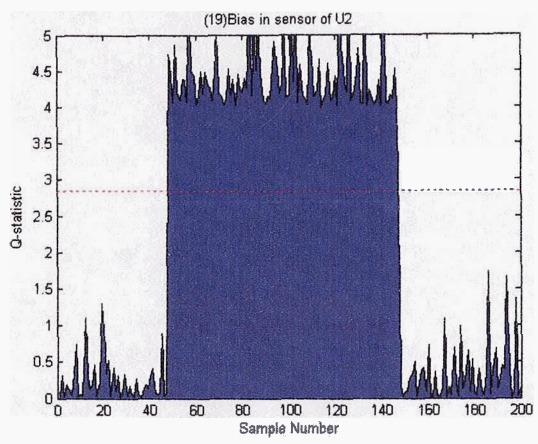


Figure 5.8 Q-statistic plots with 99% confidence limit for fault intensity of 0.2



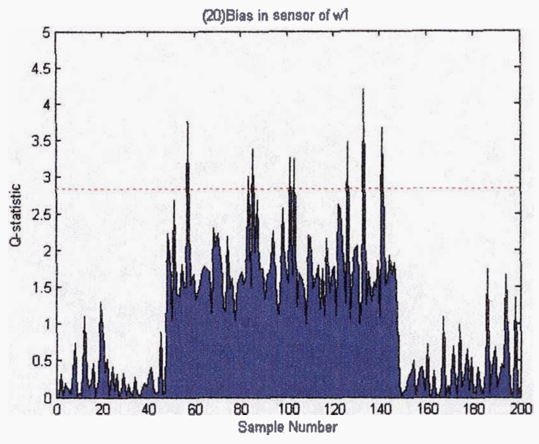


Figure 5.9 Q-statistic plots with 99% confidence limit for fault intensity of 0.2

5.3 Proposed Approach

The technique proposed in this thesis combines modern data mining techniques and PCA in an integrated framework for fault detection and identification of multivariate processes. The system can quickly and intelligently process huge amounts of data without requiring considerable computational effort for complex systems.

The proposed model of the integrated framework consists of the following components as shown in Figure 5.10:

1. Multivariate process.
2. Data Repository.
3. Data pre-processing Component.
4. Data mining component (decision tree).

Data from dynamic processes is inherently dynamic. This implies that the relationships in the data are subject to change. Therefore, any system that supports decision-making based on these data should dynamically update the models to reflect current states of the process. Otherwise, users run the risk of making decisions on process data that does not reflect the true characteristic in the current environment. In the proposed system, the data mining component is responsible for the maintenance of the classification model of the system and must constantly evaluate these models based on new data.

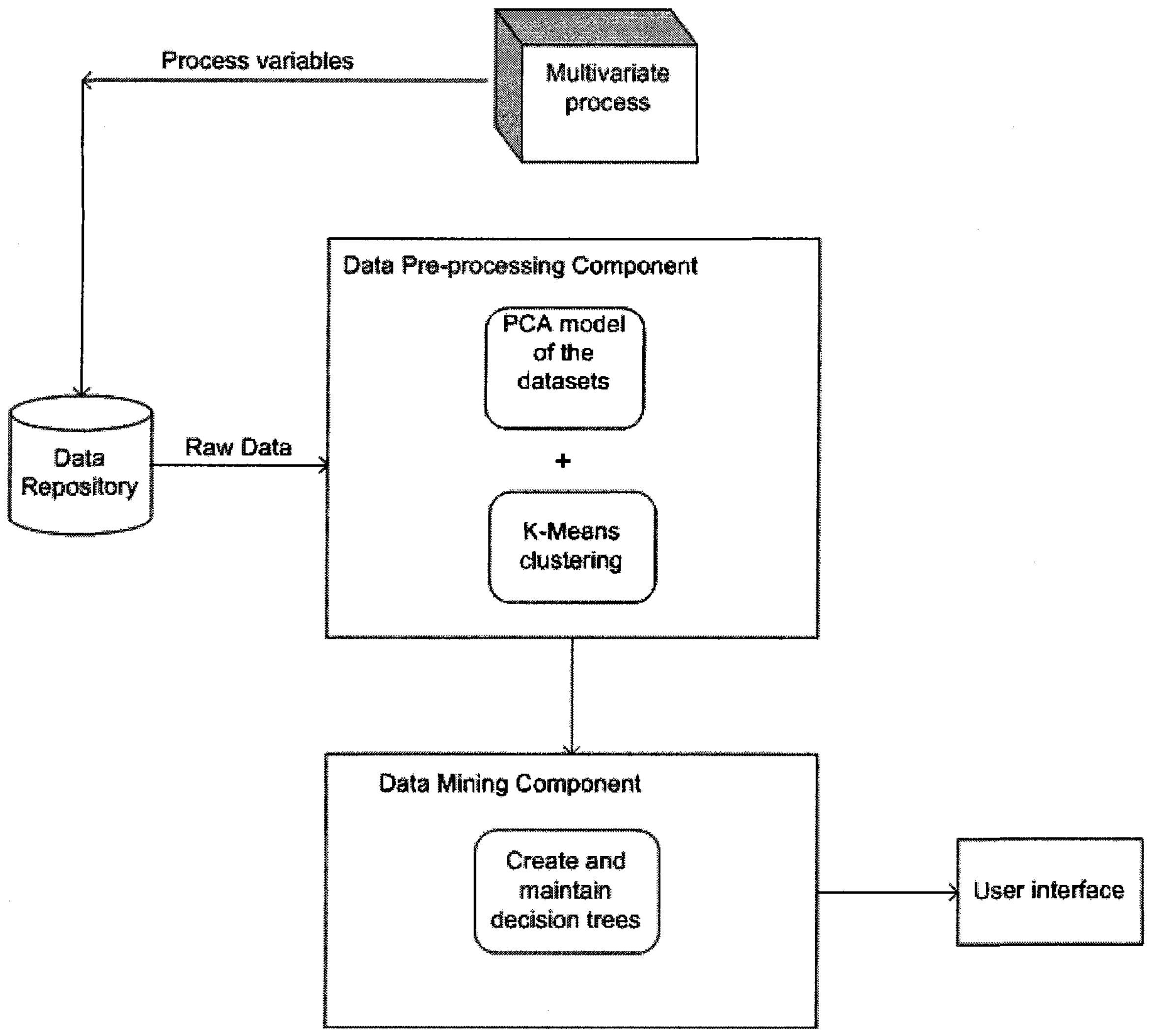


Figure 5.10 Model of the integrated framework

The data repository stores multitudes of data from measured process variables. All data mining and machine learning techniques rely heavily on the availability of sufficient volumes of "good" data to develop models of processes in sufficient detail to diagnose the system faults. It is critical to have a data-cleaning step to process conditions such as data loss in transmission due to a malfunctioning data collection device. An additional data-preprocessing step is to normalize all raw data to a common scale to allow for further processing. All necessary data-preprocessing steps are handled by "Data pre-processing component" before building PCA model from data sets.

The system needs to learn the patterns that have historically led to failure by using data mining techniques. Data mining component is responsible for creation and maintenance of the classification models associated with the process. Pre-processed and conceptualized data are fed to the data mining component, and based on these training data the decision tree is trained and stored. The result of the training process is a classification model of the system for the detection and identification of future faults. Details of building a decision-tree-based classification model of the three-tank system are given in section 5.4.3.

Data pre-processing component used in the integrated data mining framework was built in MATLAB® version 7.0 development environment. Then, it has been integrated with the decision tree building application developed by Quinlan [26]. Concept formation technique from continuous process measurements using PCA and *k*-Means clustering was introduced in Chapter 3; and C4.5 algorithm used to create decision trees for classification was described in detail in Chapter 4.

5.4 Fault Detection Using Data Mining Framework

5.4.1 Training Database Preparation

Preparing the data mining framework for fault detection requires the training of the decision tree component. A training database for the three-tank system was generated by simulating the process via Simulink® in Matlab® 7. Measurements of the 13 process variables in Table 5.1 were included in the database.

The training database was generated in the following manner. For each consecutive 1200-second period, the mode of operation (normal or faulty) to be simulated was chosen randomly using a uniform random number generator. The normal and faulty operations occurred equally frequently. If the mode of operation selected was a faulty operation, a fault was generated using the fault number following the order in Table 5.2. Then, the selected fault was introduced into the system at 1000 seconds and lasted for 200 seconds. Dynamic responses of each process variable were recorded for 200 seconds when the process was operating in fault. Faults occurred one at a time (no simultaneous faults) and for the same length of time. The same intensity level was used for each fault type during the preparation of the training data sets.

Using this simulation method, four hundred data sets were generated. For each data set, the thirteen variables were recorded as a dynamic trend consisting of 200 sampling points. Therefore, for each variable the data size is a matrix of 400 (number of data sets) $\times$ 200 (number of data points representing a dynamic trend).

Test data cases were also created to test the effectiveness of the framework for fault detection. In order to create the test database, another set of observations was generated by simulating the process for each fault listed in Table 5.2. The test database contains 20 observations (one for each fault case); and for each observation in the test database, 200 sample points were recorded as in the training cases. Therefore, for each variable the data size in the test database is a matrix of 20 $\times$ 200.

5.4.2 Data Pre-processing and Concept Formation

Principal component analysis is applied to the data matrix of 400 $\times$ 200 $\times$ 13 which contains the data cases from training database. The first two PCs from the PCA model are used to replace the dynamic trends of the process variables for subsequent *k*-Means clustering.

First two PCs of each process variable are plotted in a two-dimensional PCA plane. This reduces the dimension of the time series process data while retaining its essential character. The *k*-Means clustering technique is then applied to group PCs into clusters in this two-dimensional plane. Figures 5.11, 5.12 and 5.13 show the results of such clustering for concept formation on a two dimensional PCA plane. This permits a dynamic trend to be abstracted as a concept such as variable h2 in Cluster 3.

After the concept formation has been done on training cases, test database of 20 cases is pre-processed by applying the PCA. The two PCs of each process variables in the test database are projected onto the same PCA plane used for the concept formation of the training set. Each PCA plot from the test database then become a member of the cluster to which it is in closest proximity. PCs from the test data set are shown in red in Figures 5.11, 5.12 and 5.13.

Figure 5.11(a) shows the plotting of the first two principal components of the variable Dp1. It shows that dynamic trends of Dp1 for 400 training cases are grouped into three clusters. This means that the dynamic trends of the variable Dp1 are conceptualized into a value space of three, Cluster 1, Cluster2, and Cluster3. PCs of the same variable from 20 test cases are also partitioned into these clusters as illustrated by red dots. This technique lets us extract concept from dynamic trend signals of any given unknown (or test) data set.

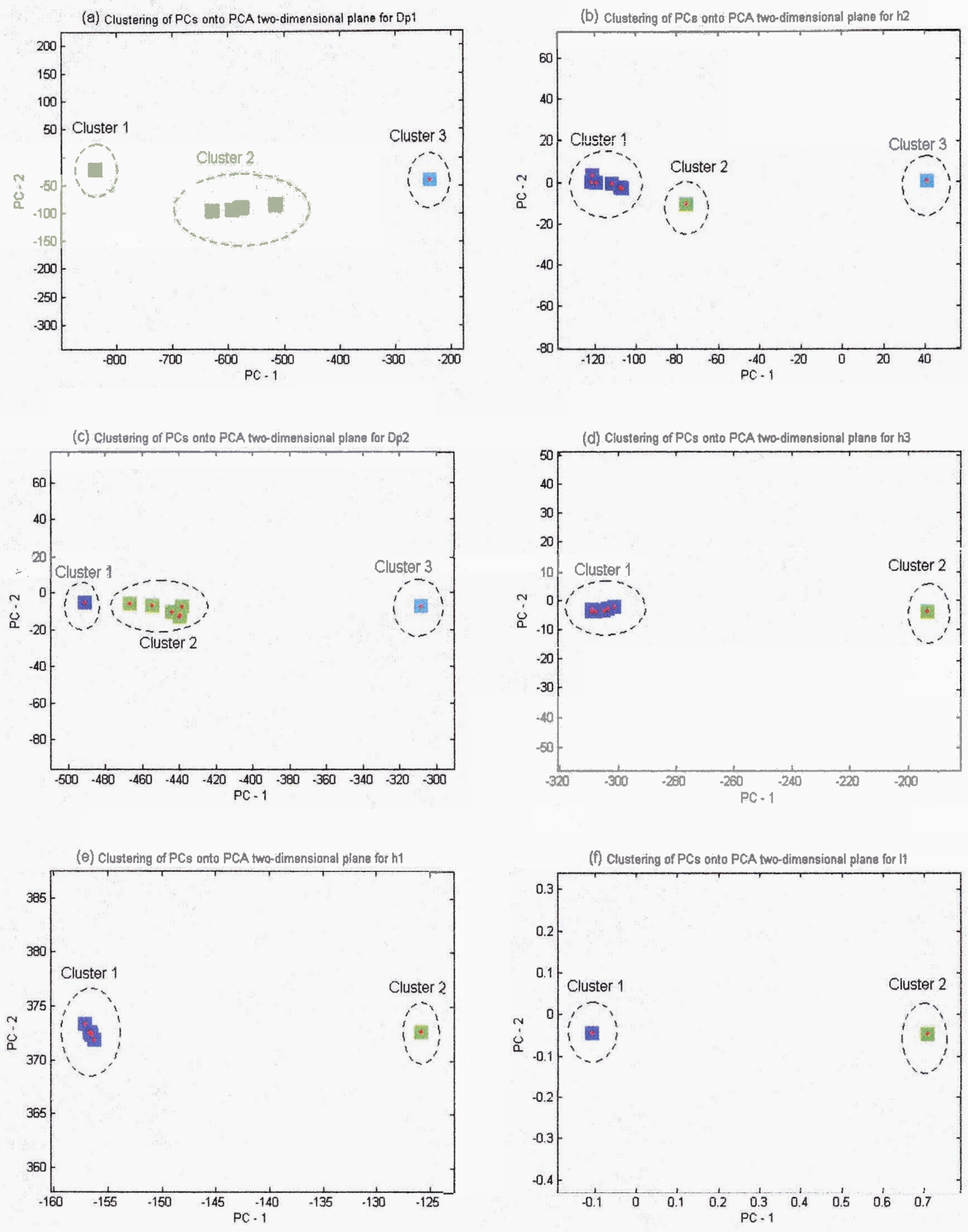


Figure 5.11 Results of k-Means clustering of PCs on PCA plane

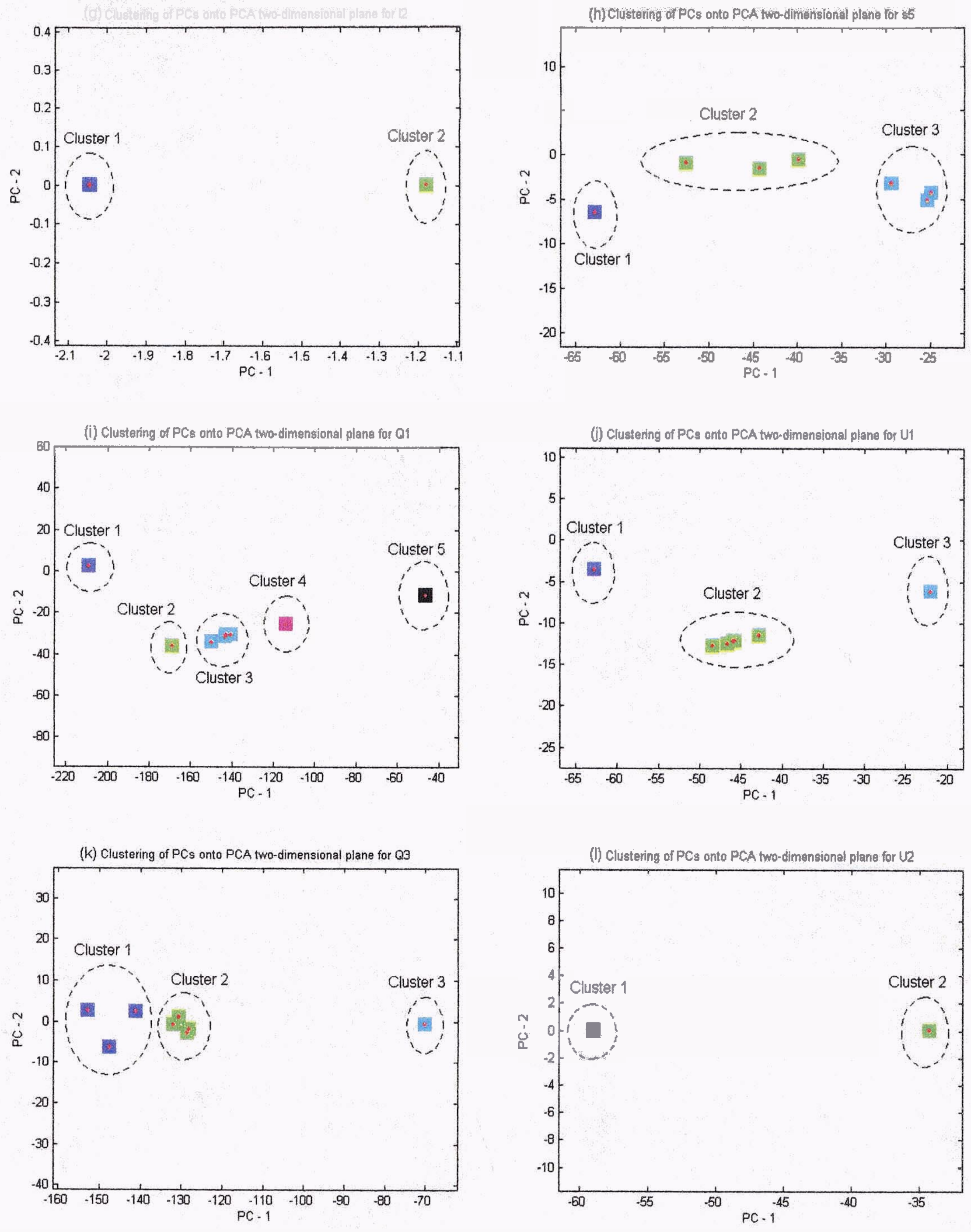


Figure 5.12 Results of k-Means clustering of PCs on PCA plane

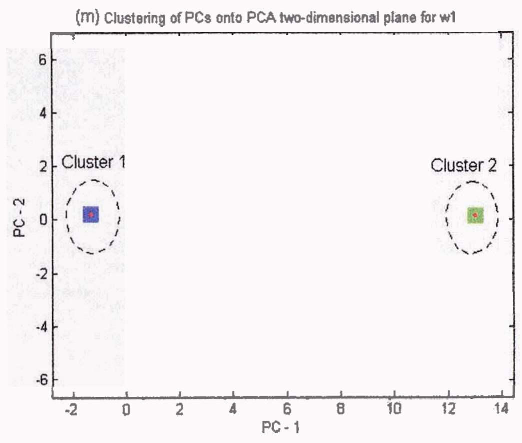


Figure 5.13 Results of k-Means clustering of PCs on PCA plane

5.4.3 Classification

Once the concept formation from dynamic trend signals is completed, the next step is to learn to generate knowledge correlating operational modes and extracted concepts.

After the data sets are pre-processed and conceptualized using the steps explained above, the output of the concept formation process is sent to the decision tree component. This requires generating a control file as shown in Table 5.4 to be used by the decision tree generation tool.

Table 5-4 Structure of the control file used by the decision tree generation tool

Variable name	Value space
h1	Cluster 1, Cluster 2
h2	Cluster 1, Cluster 2, Cluster 3
h3	Cluster 1, Cluster 2
Q1	Cluster 1, Cluster 2, Cluster 3, Cluster 4, Cluster 5
Dp1	Cluster 1, Cluster 2, Cluster 3
U1	Cluster 1, Cluster 2, Cluster 3
I1	Cluster 1, Cluster 2
w1	Cluster 1, Cluster 2
Q3	Cluster 1, Cluster 2, Cluster 3
s5	Cluster 1, Cluster 2, Cluster 3
Dp2	Cluster 1, Cluster 2, Cluster 3
U2	Cluster 1, Cluster 2
I2	Cluster 1, Cluster 2

Conceptualized process variables in each data set are expressed in a file as illustrated in Table 5.5. Each row in Table 5.5 shows the cluster number which each process variable is grouped into. Class label value is used for training data set only as it is used to train the decision tree. For the test cases; this column will be left blank since predicting the class labels for unknown cases is our ultimate goal in building decision tree. This data structure together with the control file shown in Table 5.4 is used to generate the decision tree classification model.

Table 5-5 Partial data structure used to train the decision tree

h1	h2	h3	Q1	Dp1	U1	I1	w1	Q3	s5	Dp2	U2	I2	Class Label
1	1	2	5	2	2	1	1	1	2	1	1	2	biash3
2	1	1	1	1	1	1	1	1	3	1	1	1	biash1
1	3	1	4	2	2	2	1	2	3	1	1	1	biash2
2	1	1	1	1	1	1	1	1	3	2	2	1	biash1
1	3	1	1	2	2	1	1	1	3	2	2	1	biash2
1	1	1	3	2	2	2	2	2	3	2	1	1	biasQ1
1	1	1	1	3	3	1	2	3	3	2	1	1	biasDp1
1	1	1	2	2	3	2	1	3	2	2	1	1	biasI1

A decision tree is generated and saved using known data cases from the training database. Once the decision tree has been created using 400 training cases, test cases can be classified using this decision tree. In other words, class labels of each data case in the test database can be predicted to detect faults if there is any. The decision tree developed from the training database is shown in Figure 5.14.

```
Q3 = 1: faulty (10)
Q3 = 3: faulty (31)
Q3 = 2:
:...h2 = 3: normal (0)
    h2 = 1: faulty (20)
    h2 = 4: faulty (10)
    h2 = 2:
    :...Q1 in 1,2: normal (0)
        Q1 = 4: faulty (11)
        Q1 = 5: faulty (10)
        Q1 = 3:
        :...I1 = 2: faulty (10)
            I1 = 1:
            :...w1 = 2: faulty (10)
                w1 = 1:
                :...U2 = 2: faulty (10)
                    U2 = 1:
                    :...I2 = 2: faulty (10)
                        I2 = 1:
                        :...h3 = 3: normal (0)
                            h3 = 2: faulty (10)
                            h3 = 1:
                            :...Dp1 = 1: normal (0)
                                Dp1 = 3: faulty (10)
                                Dp1 = 2:
                                :...U1 = 1: normal (0)
                                    U1 = 3: faulty (10)
                                    U1 = 2:
                                    :...Dp2 = 1: normal (0)
                                        Dp2 = 2: normal (228/30)
                                        Dp2 = 3: faulty (10)
```

Figure 5.14 Decision tree generated for training database with fault intensity 0.2

```
Evaluation on test data (20 cases):

          Decision Tree
        ----------------
        Size      Errors

          15    3(15.0%)   <<


         (a)   (b)    <-classified as
        ----  ----
                      (a): class normal
           3    17    (b): class faulty
```

Figure 5.15 Evaluation results of the decision tree on 20 test cases

The numbers in parentheses appearing after a leaf indicates the number of the training cases associated with each leaf and the number of them misclassified by the leaf. For example, the leaf (Dp2 = 2) with class label *normal* has 228 cases associated with it and it misclassifies 30 faulty cases out of 228 as normal.

The decision tree shown in Figure 5.14 misclassifies 30 of the 400 training cases, which is equivalent to 7.5% error rate. Once the tree has been built, the file of 20 test cases is processed and each case classified by the tree. The decision tree misclassifies only 3 test cases out of 20 as normal, achieving a success rate of 85%. Under the same conditions, the traditional PCA method had a success rate of 70%.

The decision tree component also generates a *confusion matrix* as part of the output on the test cases, showing how the misclassifications were distributed. Figure 5.15 shows the confusion matrix created on the test cases. There are 20 test cases of class faulty, 17 of which are correctly classified as faulty while 3 are misclassified as normal. Table 5.6 shows all data cases in the test database along with the predicted class label for each case. The proposed framework cannot detect the following faults with intensity 0.2: *clog in branch with P1*, *clog in branch with P2*, and *increased friction in P1*.

Table 5-6 Classification results for the test database with fault intensity 0.2.

Fault No	Fault Description	Given Class	Predicted Class
1	bias in sensor of h1	faulty	faulty
2	bias in sensor of h2	faulty	faulty
3	bias in sensor of h3	faulty	faulty
4	leak in R1	faulty	faulty
5	clog in branch with V1	faulty	faulty
6	clog in branch with V2	faulty	faulty
7	clog in branch with V4	faulty	faulty
8	**clog in branch with P1**	**faulty**	**normal**
9	**clog in branch with P2**	**faulty**	**normal**
10	**friction in P1**	**faulty**	**normal**
11	clog in branch with V5	faulty	faulty
12	bias in sensor of Q1	faulty	faulty
13	bias in sensor of Dp1	faulty	faulty
14	bias in sensor of Dp2	faulty	faulty
15	bias in sensor of I1	faulty	faulty
16	bias in sensor of U1	faulty	faulty
17	bias in sensor of Q3	faulty	faulty
18	bias in sensor of I2	faulty	faulty
19	bias in sensor of U2	faulty	faulty
20	bias in sensor of w1	faulty	faulty

The framework was also tested using the process data generated by running the three-tank system with faults at intensity levels 0.5 and 1. The results of these simulations are given in Table 5.7 and Table 5.8, respectively. All misclassified fault types are shown in bold font in all tables.

As seen in Table 5.7, when the intensity of the faults is increased to 0.5, the performance of the framework does not improve. These represent very similar –if not the same- results, that previously seen in the simulation performed with the intensity level of 0.2.

When we increase the fault intensity to 1, the system has a success rate of 90%: only 2 out of 20 faulty cases are misclassified as normal (see Table 5.8).

Table 5-7 Classification results for the test database with fault intensity 0.5.

Fault No	Fault Description	Given Class	Predicted Class
1	bias in sensor of h1	faulty	faulty
2	bias in sensor of h2	faulty	faulty
3	bias in sensor of h3	faulty	faulty
4	leak in R1	faulty	faulty
5	clog in branch with V1	faulty	faulty
6	clog in branch with V2	faulty	faulty
7	clog in branch with V4	faulty	faulty
8	**clog in branch with P1**	**faulty**	**normal**
9	**clog in branch with P2**	**faulty**	**normal**
10	**friction in P1**	**faulty**	**normal**
11	clog in branch with V5	faulty	faulty
12	bias in sensor of Q1	faulty	faulty
13	bias in sensor of Dp1	faulty	faulty
14	bias in sensor of Dp2	faulty	faulty
15	bias in sensor of I1	faulty	faulty
16	bias in sensor of U1	faulty	faulty
17	bias in sensor of Q3	faulty	faulty
18	bias in sensor of I2	faulty	faulty
19	bias in sensor of U2	faulty	faulty
20	bias in sensor of w1	faulty	faulty

Table 5-8 Classification results for the test database with fault intensity 1.

Fault No	Fault Description	Given Class	Predicted Class
1	bias in sensor of h1	faulty	faulty
2	bias in sensor of h2	faulty	faulty
3	bias in sensor of h3	faulty	faulty
4	leak in R1	faulty	faulty
5	clog in branch with V1	faulty	faulty
6	clog in branch with V2	faulty	faulty
7	clog in branch with V4	faulty	faulty
8	clog in branch with P1	faulty	faulty
9	**clog in branch with P2**	**faulty**	**normal**
10	**friction in P1**	**faulty**	**normal**
11	clog in branch with V5	faulty	faulty
12	bias in sensor of Q1	faulty	faulty
13	bias in sensor of Dp1	faulty	faulty
14	bias in sensor of Dp2	faulty	faulty
15	bias in sensor of I1	faulty	faulty
16	bias in sensor of U1	faulty	faulty
17	bias in sensor of Q3	faulty	faulty
18	bias in sensor of I2	faulty	faulty
19	bias in sensor of U2	faulty	faulty
20	bias in sensor of w1	faulty	faulty

5.5 Fault Identification using Data Mining Framework

5.5.1 Training Database Preparation

Preparing the data mining framework for fault identification also requires a fault training database. In this case study, a database of 500 data sets was obtained by carrying out various tests on the simulator. Each data set consists of thirteen variables, which are listed in Table 5.2. Each variable represents a dynamic trend consisting of 200 sample points. Therefore, for each variable the data size is a matrix of 500 (number of data sets) $\times$ 200 (number of data points representing a dynamic trend).

The intensity of the fault in the training database was picked randomly using a uniform random number generator that ranged from 0.5 to 1. The simulation studies have shown that the performance of the clustering component degrades quickly for the faults whose severity is lower than 0.5. Since this affects the performance of the framework as a whole, only faults whose severity is between 0.5 and 1 are introduced into the system during the simulation. Possible improvements that can be done on the clustering component are discussed in the Chapter 6.

Once the relative fault size was selected, the simulation ran for 1200 seconds for each of the 20 fault types before a new fault size was selected. All fault lengths were 200 seconds in duration; therefore, the process was given 1000 seconds to return to the original steady state before each period of faulty operations started. Using this simulation method, faults in the training database were constructed with various intensity levels, which makes the training database resemble the historical database of a real life multivariate system as much as possible.

5.5.2 Data Pre-processing and Concept Formation

The training database is preprocessed to prepare the time-series process data for subsequent concept formation step.

First, principal component analysis (PCA) is applied to a matrix of 500 (number of data cases) $\times$ 200 (number of sample points recorded for each process variable) $\times$ 13 (number of process variables) to reduce the dimension of the data set. The first two PCs of each process variable are plotted on a two-dimensional PCA plane to replace the dynamic trends. Then, *k*-Means clustering algorithm is applied to conceptualize variables into clusters. Each process variable takes discrete values from a region (or cluster) of the two-dimensional PCA plane.

Figures 5.16, 5.17 and 5.18 shows the results of *k*-Means clustering applied to the PCs of each process variable for concept formation.

5.5.3 Classification

After pre-processing the training database and extracting the concept from continuous process variables, the results were sent to the decision tree component to identify the unknown faults. 400 cases in the training database were used to train the decision tree while 100 cases were kept aside to test the effectiveness of the data mining framework for fault identification with unknown fault records. Test cases contained faults with various intensity levels as the faults in the training cases.

The decision tree built by the decision tree component is shown in Figure 5.19. The root node is Q3, which is the flow flowing to tank R3. This indicates that it is the most important variable that distinguishes operational modes representing the 400 training cases. Once the decision tree is built and saved, test cases can be processed by the tree to identify the fault type.

The confusion matrix, which shows the results of the classification on the test cases, is given in Figure 5.20. The confusion matrix is the final part of the output generated by the framework and it shows how the misclassifications were distributed. There are 100 test cases in the test database, 10 of which are misdiagnosed. There are 5 test cases representing each faulty operation and only the test cases generated from fault types 'clog in branch with P1' and 'increased friction in DC motor of P1' are misclassified as 'clog in branch with P2'.

This test shows that data mining based fault identification system performs accurate identification of single faults in the tree-tank system. It can be seen from the final output of the data mining component that the proposed data mining framework can identify the unknown faults occurred in the system with an accuracy of 90%.

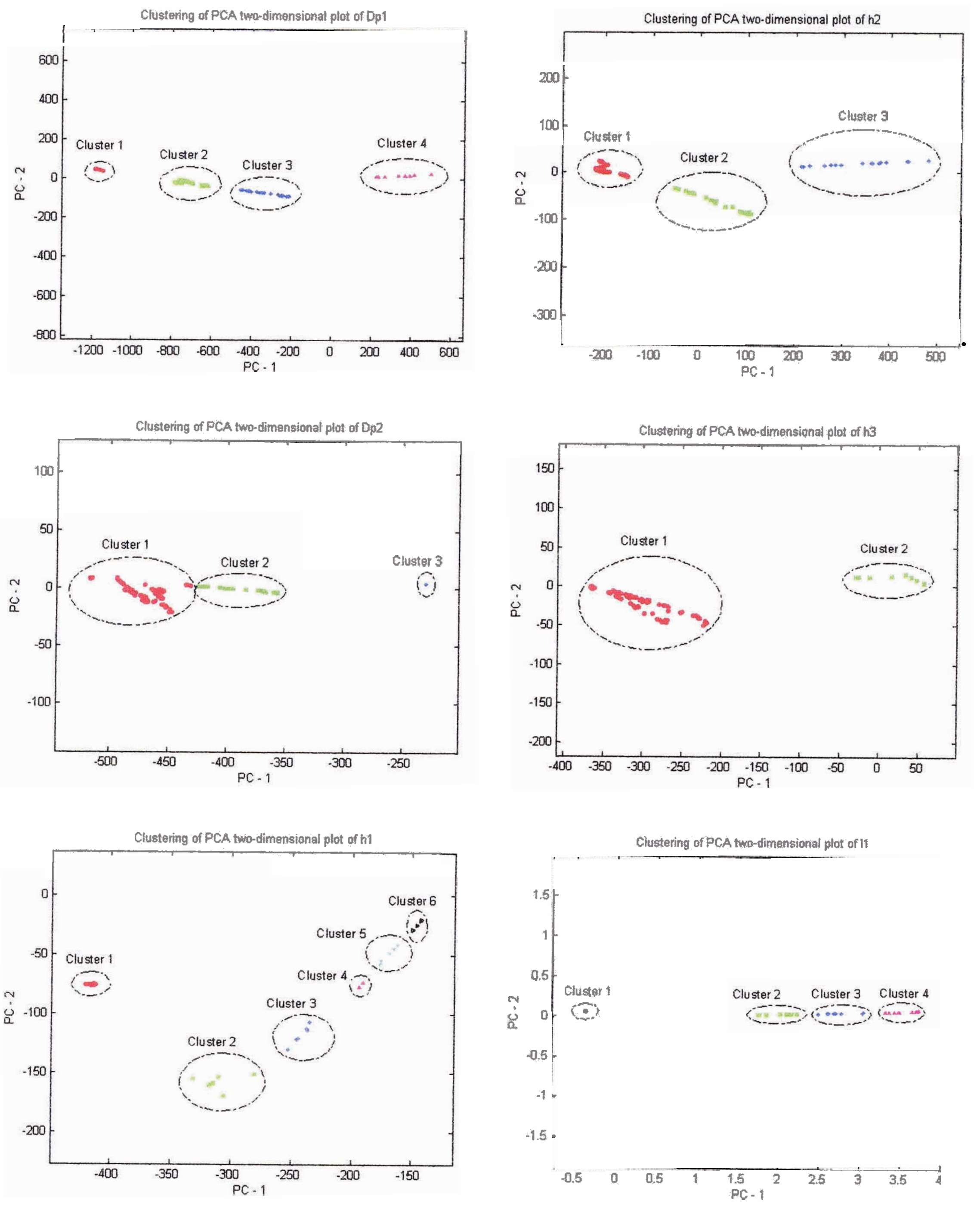


Figure 5.16 Results of k-Means clustering of PCs of process variables on two-dimensional plane.

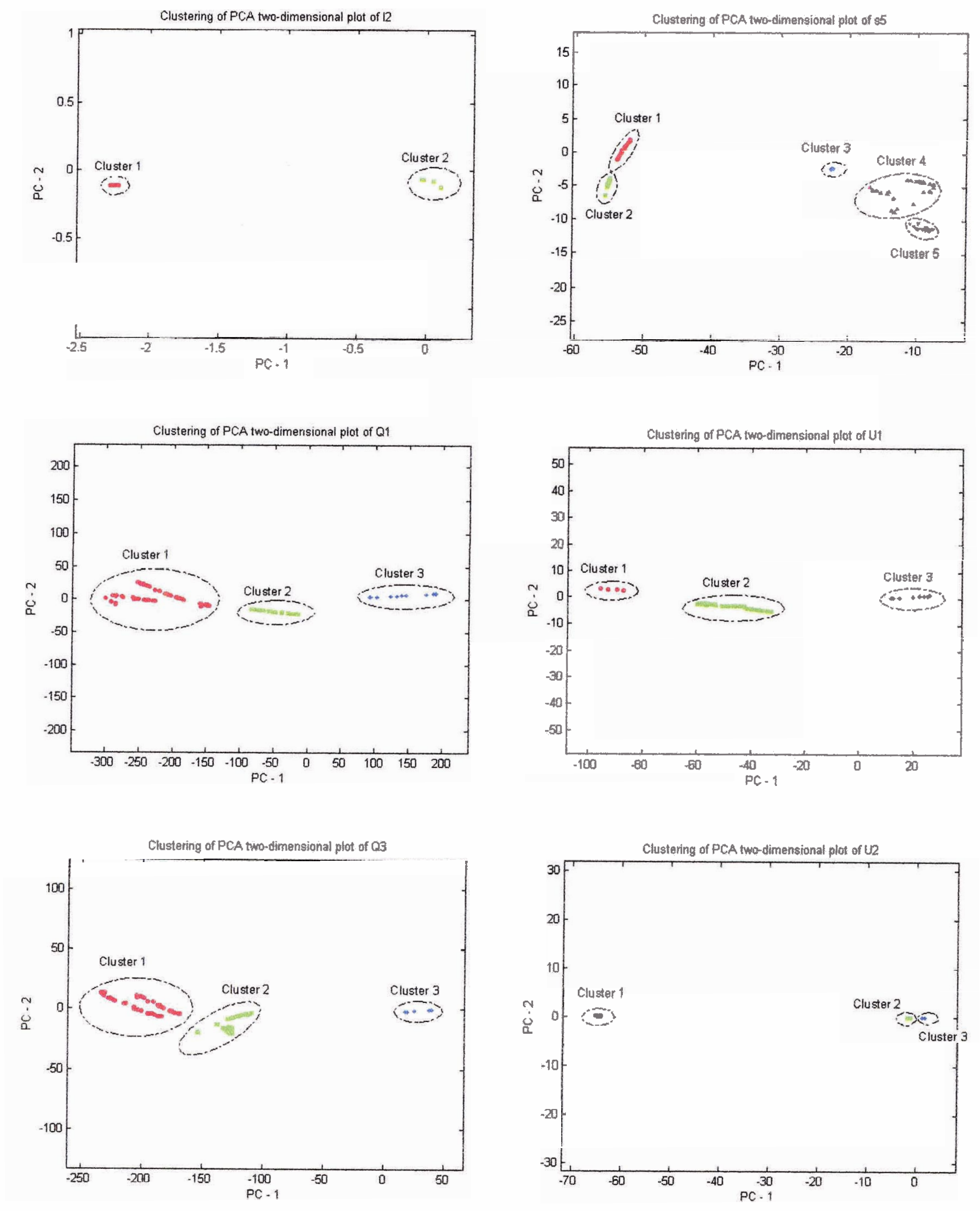


Figure 5.17 Results of k-Means clustering of PCs of process variables on two-dimensional plane.

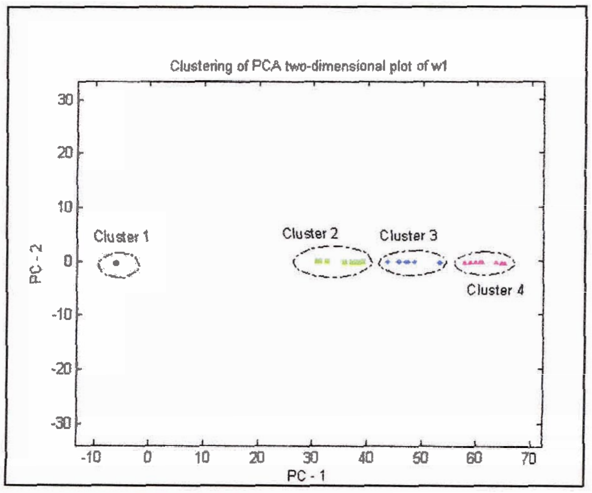


Figure 5.18 Results of k-Means clustering of PCs of process variable w1

```
Q3 = 3: biasQ3 (20)
Q3 = 1:
:...h2 = 3: clogV2 (0)
:   h2 = 1: clogV2 (20)
:   h2 = 2: clogV4 (20)
Q3 = 2:
:...Q1 = 2: leakR1 (20)
    Q1 = 3: biasQ1 (20)
    Q1 = 1:
    :...U1 = 1: biash1 (20)
        U1 = 3: biasU1 (20)
        U1 = 2:
        :...s5 in 1,5: biash3 (0)
            s5 = 4: clogV1 (20)
            s5 = 2:
            :...h3 = 1: clogV5 (20)
            :   h3 = 2: biash3 (20)
            s5 = 3:
            :...I2 = 2: biasI2 (20)
                I2 = 1:
                :...h2 = 2: clogP2 (0)
                    h2 = 3: biash2 (20)
                    h2 = 1:
                    :...Dp2 = 2: clogP2 (0)
                        Dp2 = 3: biasDp2 (20)
                        Dp2 = 1:
                        :...Dp1 in 1,3: clogP2 (0)
                            Dp1 = 4: biasDp1 (20)
                            Dp1 = 2:
                            :...U2 = 2: biasU2 (1)
                                U2 = 3: biasU2 (19)
                                U2 = 1:
                                :...I1 = 2: biasI1 (4)
                                    I1 = 3: biasI1 (7)
                                    I1 = 4: biasI1 (9)
                                    I1 = 1:
                                    :...w1 = 1: clogP2 (60/40)
                                        w1 = 2: biasw1 (4)
                                        w1 = 3: biasw1 (7)
                                        w1 = 4: biasw1 (9)
```

Figure 5.19 Decision tree generated from 400 training cases.

5.5.4 Previous Work and Discussion

In a recent study, Li [5] has developed an automated framework for fault diagnosis based on nonlinear principal component analysis (NLPCA) neural network, and tested it on the three-tank benchmark system. Her approach was successful to diagnose single faults only if the training data set used to train the neural network contained the faults with the same intensity level as in the test data set. She has trained the neural network with the faults created at two severity levels, 0.1 and 0.5. The framework diagnoses the faults correctly if their intensity is at 0.1 or 0.5. When a fault with different intensity level from those in the training set occurs, the performance degrades significantly. The success rate of the framework degrades to 55% in some cases for the faults whose intensity varies from the training set.

Neural network approach requires extensive amount of training data cases to successfully represent a complex real life system and it is computationally expensive. A major limitation of neural-network based approach is that it gives predictions but not causal and qualitative explanations. This means that in process operational decision support, it is not able to indicate to operators what variables are responsible for the diagnosed fault and provides no clues for operational adjustment [2, 29].

The proposed data-mining framework addresses this limitation of the neural network based approach by using a training database that contains faults whose intensity was randomly picked. The case study has showed the system's ability to accurately diagnose randomly generated faults. The proposed approach also gives causal explanations of various faults in the form of decision trees.

5.6 Conclusions

In this chapter, the implementation of data mining framework was tested on the tree-tank benchmark system. First, traditional PCA based monitoring technique, Q-statistic, was used to detect single faults occurring in the system. The highest success rate that was achieved by this technique was 80% with fault intensity 1. Then, the framework has been compared to the Q-statistic technique to demonstrate the fault detection capability of the proposed approach. The test results show that proposed approach outperforms the traditional PCA based fault detection technique in all fault intensity levels.

Second, the framework was used as fault identification system to correctly identify single faults. In this case study, the training database was created with faults whose intensity was selected randomly. Test cases were also created in the same manner to demonstrate the fault identification capability of the framework regardless of the fault size used in the training data set. For the three-tank benchmark system, the results show that the proposed data mining framework identifies faults accurately in many cases.

```
Evaluation on test data (100 cases):

        Decision Tree
      -----------------
      Size      Errors

        23    10(10.0%)

(a)  (b)  (c)  (d)  (e)  (f)  (g)  (h)  (i)  (j)  (k)  (l)  (m)  (n)  (o)  (p)  (q)  (r)  (s)  (t)   <-classified as
 5                                                                                                  (a):class biash3
      5                                                                                             (b):class leakR1
           5                                                                                        (c):class clogP2
                5                                                                                   (d):class clogV1
                     5                                                                              (e):class clogV2
                          5                                                                         (f):class clogV4
           5                                                                                        (g):class clogP1
           5                                                                                        (h):class frictP1
                                         5                                                          (i):class clogV5
                                              5                                                     (j):class biash1
                                                   5                                                (k):class biash2
                                                        5                                           (l):class biasQ1
                                                             5                                      (m):class biasDp1
                                                                  5                                 (n):class biasI1
                                                                       5                            (o):class biasU1
                                                                            5                       (p):class biasQ3
                                                                                 5                  (q):class biasDp2
                                                                                      5             (r):class biasI2
                                                                                           5        (s):class biasU2
                                                                                                5   (t):class biasw1
```

Figure 5.20 Confusion matrix generated by the decision tree for the test cases.

CHAPTER 6: CONCLUSIONS

In large industrial plants, modern distributed control and automatic data logging systems collect large amount of data that contain valuable information about both normal and abnormal operations. A data mining system has considerable potential in extracting knowledge from such data that can be used for fault detection and identification. The objective of this research was to exploit historical process data by applying well known data mining techniques to gain insight into the behavior of a complex multivariate process. An integrated data mining framework combining statistical methods with modern data mining techniques has been developed.

There are numerous techniques in data mining and machine learning, which have proved to be very successful. The way of making use of data depends on the type of learning: Supervised (e.g. neural-network based machine learning, decision tree) or unsupervised (e.g. clustering). Although supervised learning normally gives more accurate predictions, there are often difficulties in finding training data. However, dynamic simulators have proved to be an effective way to generate training cases. Another problem with supervised learning is that supervised training is not effective in dealing with new cases that are beyond the range of training patterns. Unsupervised learning methods would require no training data but tend to give less accurate results. In the proposed framework, both supervised and unsupervised techniques have been used to take advantage of both approaches.

The most critical step in building such data mining system was extracting concepts from multivariate time-series process data. To be able to extract knowledge from dynamic signals was the key to the success of the framework.

For this purpose, an approach using principal component analysis proposed by Wang and Li [2] was adapted. The adapted PCA method works as a visualization-based analysis tool to help the user lead the subsequent clustering process, which is an unsupervised learning. The PCA and clustering together comprise the data pre-processing step of the KDD process which proposed data mining framework employs. The output of the data pre-processing component is then fed to the decision tree component, which is a supervised learning technique.

The approach was illustrated on a three-tank benchmark system, which is a highly nonlinear multivariate process. It has been showed that the proposed data-mining based fault detection and identification scheme can detect and identify single faults very accurately.

Although the approach is well founded, there are some limitations to be addressed. One of the key assumptions made during concept extraction from dynamic trend signals is that the first two PCs can represent most of the variation of the variable. This assumption may not be true in some practical industrial processes in which case a further analysis should be run to determine appropriate number of principal components during data pre-processing step. Another limitation of this framework comes from the use of *k*-Means clustering. *k*-Means clustering is a very fast and effective technique and employed in many data mining tools, yet it cannot recognize the non-spherical clusters, which might pose some limitations during concept formation process.

Because the field of data mining and its application to fault detection and identification is so new, the possibilities for future research are enormous. A few suggestions are given here.

Evaluating clustering techniques

Future improvements to the data pre-processing component can be done in several directions. Standard *k*-Means clustering algorithm can be modified to detect the true number of clusters automatically for each variable. The end user can choose either to lead the concept extraction process iteratively by interaction with the system through user interface or to have a completely automatic KDD process.

When clusters are not neatly expressed as Gaussian noise around a central point many things can go wrong in a *k*-Means approach. Newer clustering algorithms such as CURE and CHAMELEON can be employed to be better able to handle clusters of arbitrary shapes and sizes [38]. These new clustering techniques can significantly improve the framework's overall fault detection ability even under the faults occurring at a very low intensity level.

Background Knowledge

It is also important to include the role of background knowledge and a model of the domain in the KDD process. Much of that is resident only in the mind of the domain expert, but principal component analysis technique can take advantage of formally represented knowledge in the course of fitting data to the model. The proposed system can be modified to allow the user to define a threshold to be used for finding optimum number of principal components during the PCA modeling of the process variables for concept extraction.

Graphical User Interface

This research explores the integration of a heterogeneous suite of data mining techniques in an integrated framework. The current system has been made up of several scripts that run asynchronously and requires user interaction to lead the KDD process. In order to have a better understanding of the problem domain, an effective graphical user interface can be incorporated into the system. The output generated by the decision tree component is given in text format. Data visualization techniques can be applied to the output to visualize complex decision tree structures and rule sets.

APPENDICES

Appendix 1: Q-statistic Charts for a Single Fault with Intensity 0.5

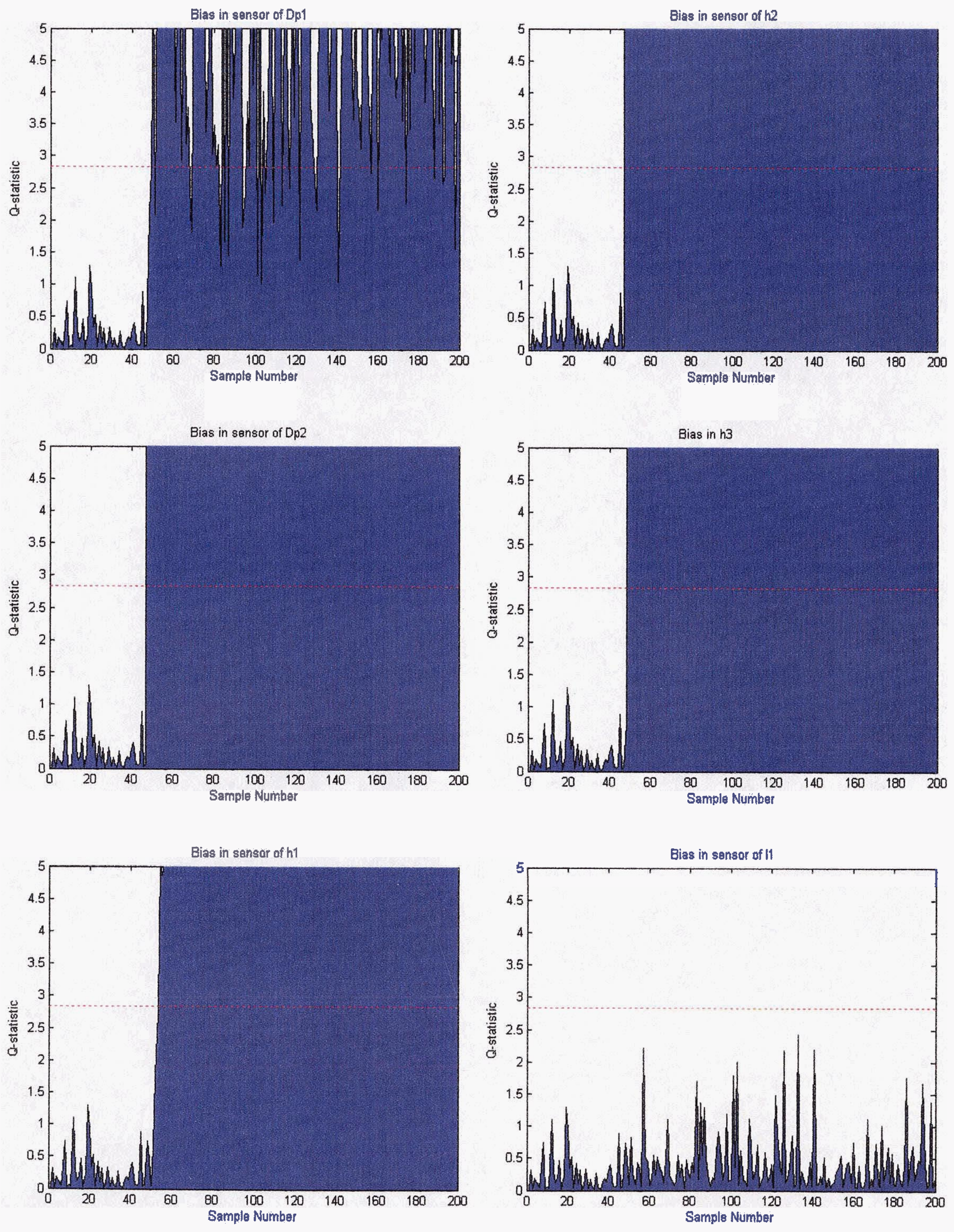


Figure A1.1 Q-statistic plots with 99% confidence limit for a single fault of intensity 0.5

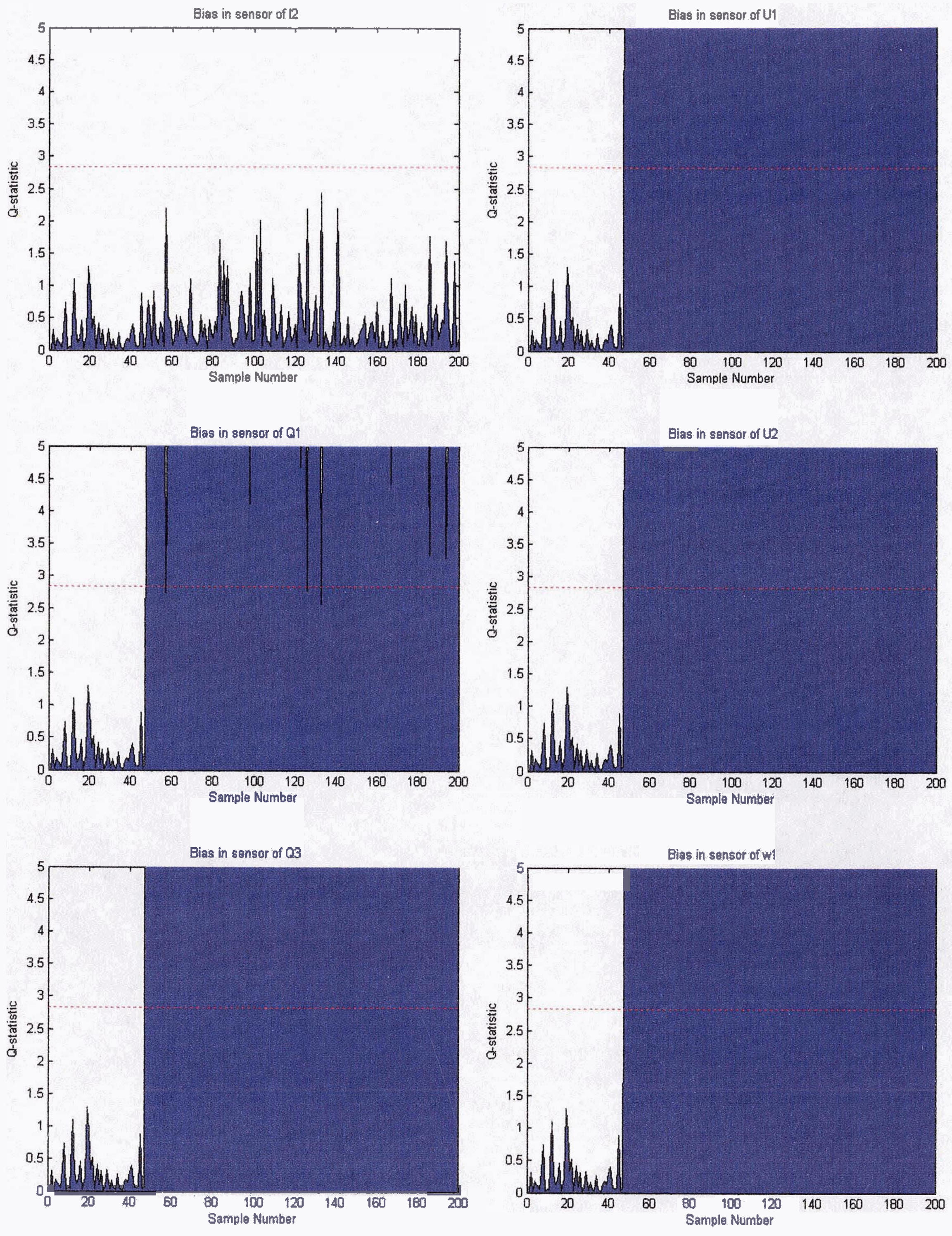


Figure A1.2 Q-statistic plots with 99% confidence limit for a single fault of intensity 0.5

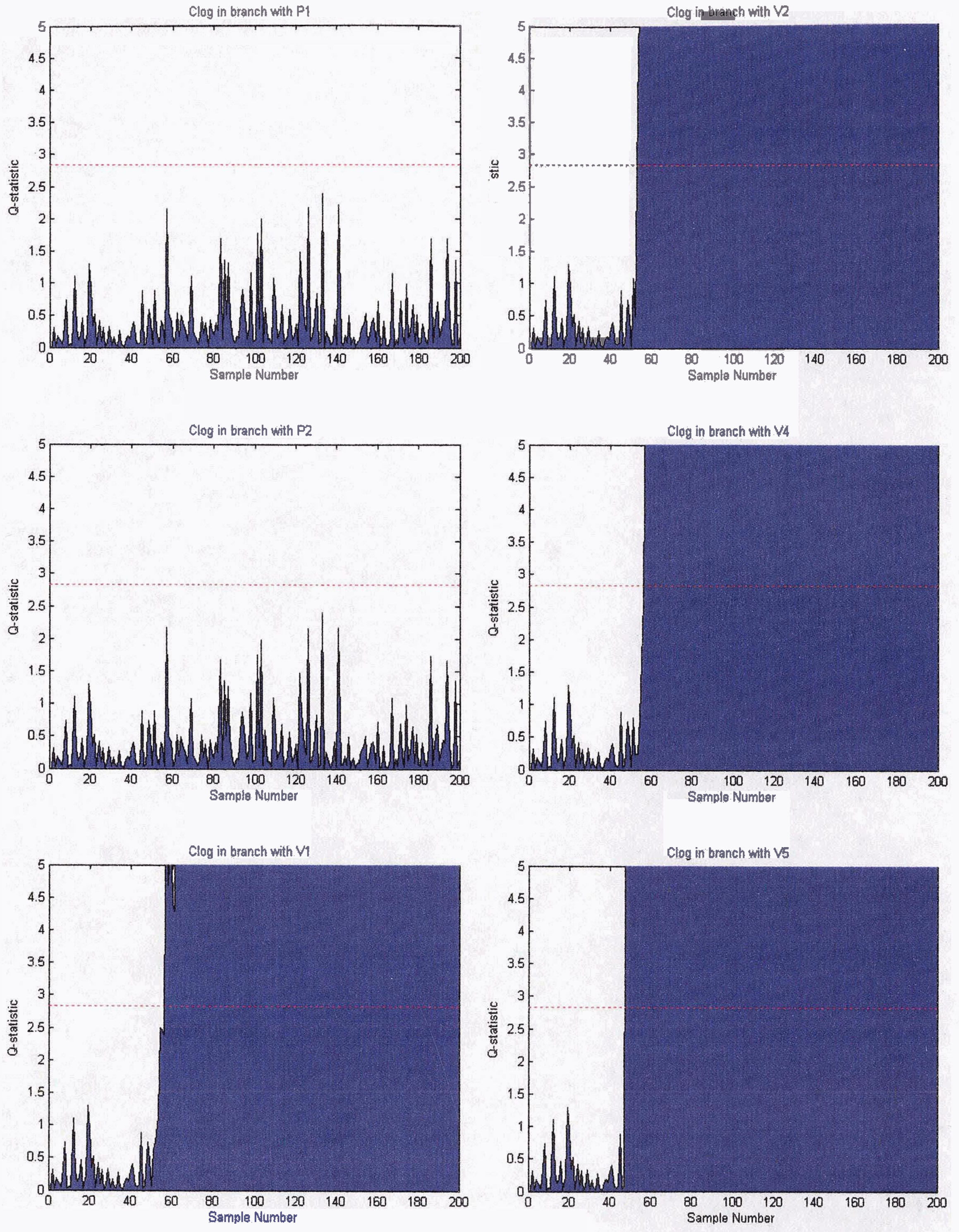


Figure A1.3 Q-statistic plots with 99% confidence limit for a single fault of intensity 0.5

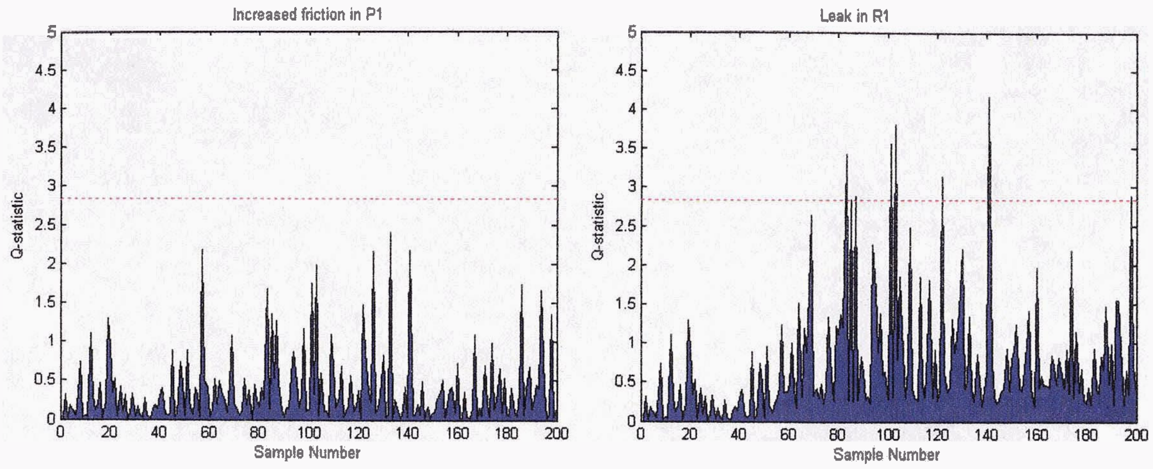


Figure A1.4 Q-statistic plots with 99% confidence limit for a single fault of intensity 0.5

Appendix 2: Q-statistic Charts for a Single Fault with Intensity 1.

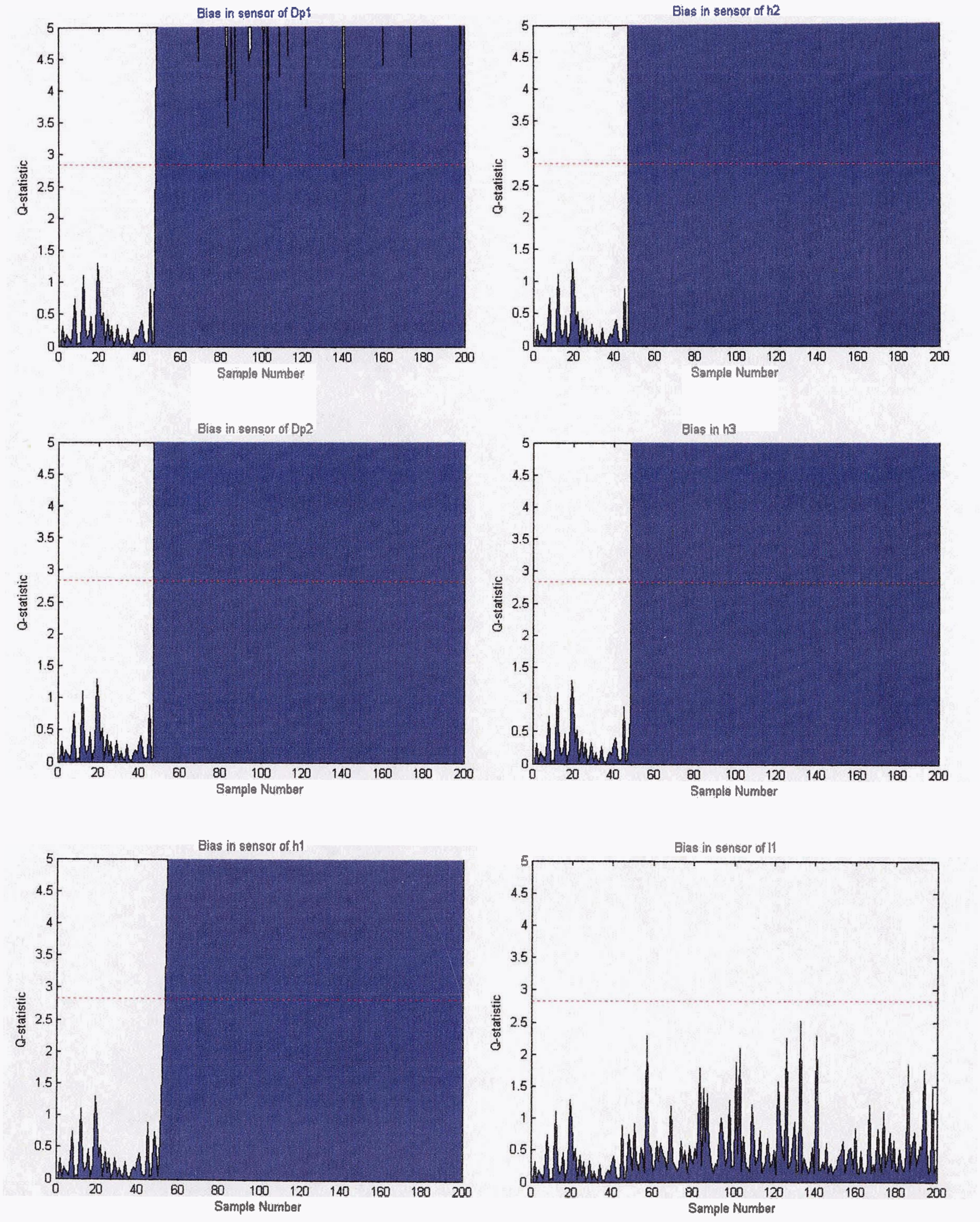


Figure A2.1 Q-statistic plots with 99% confidence limit for a single fault of intensity 1

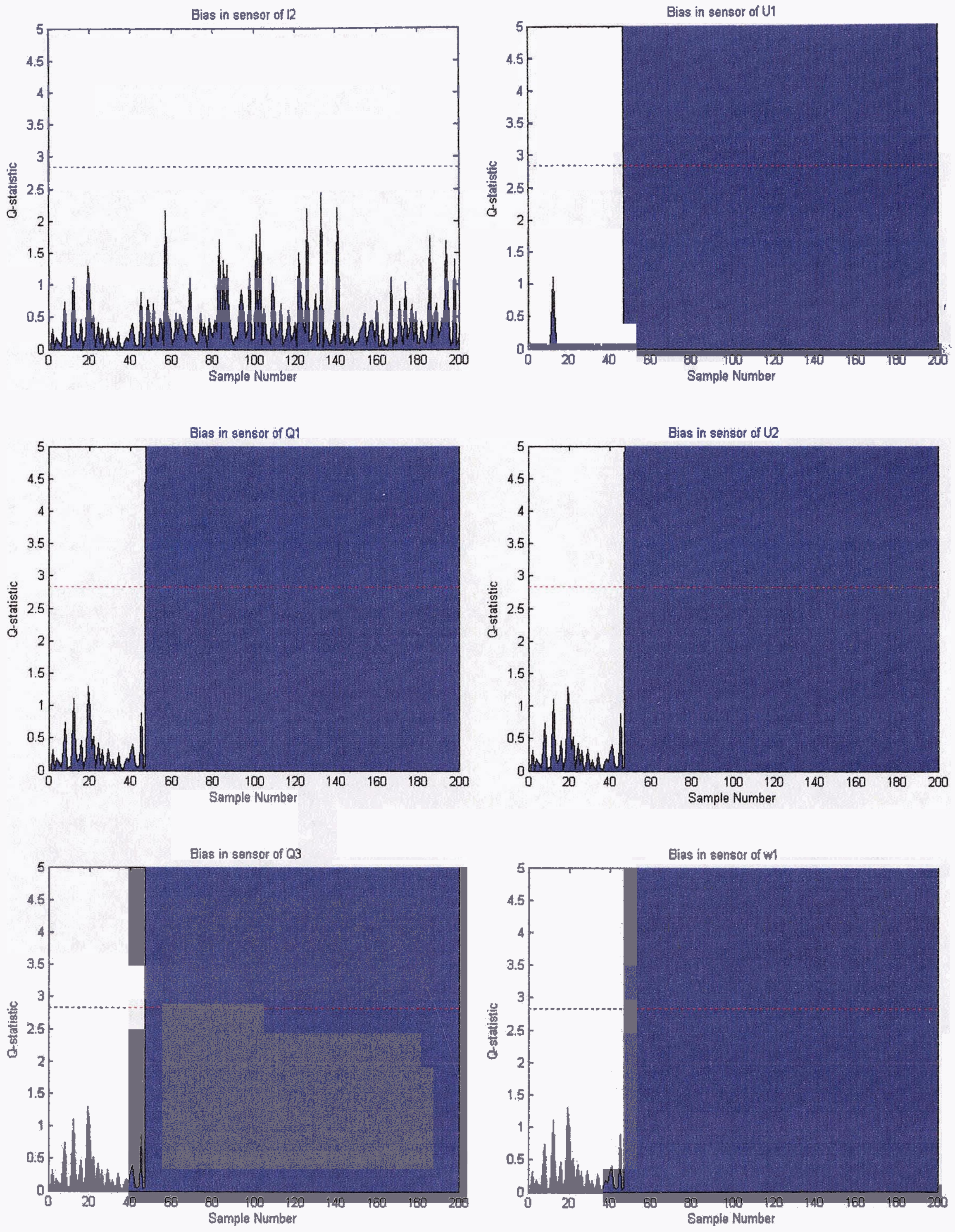


Figure A2.2 Q-statistic plots with 99% confidence limit for a single fault of intensity 1

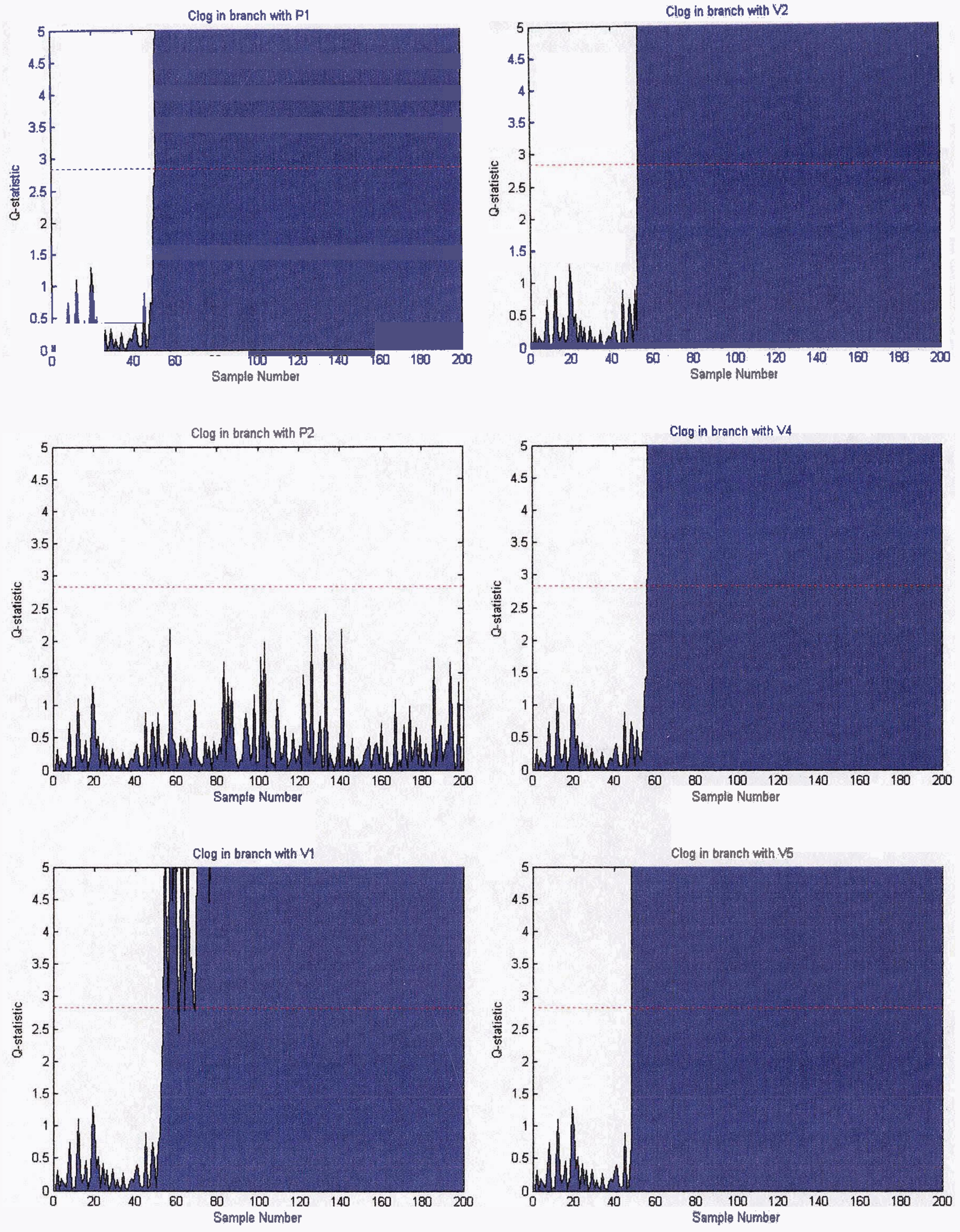
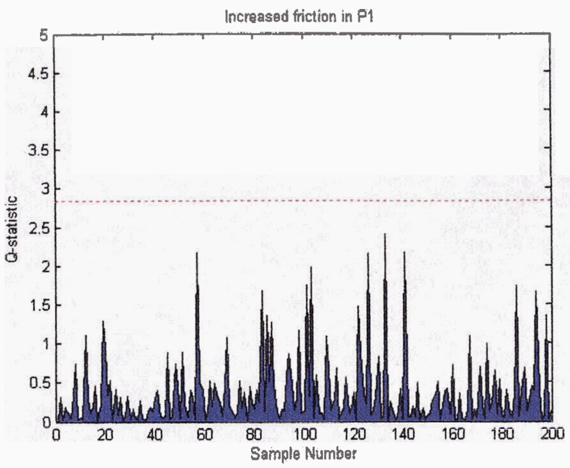


Figure A2.3 Q-statistic plots with 99% confidence limit for a single fault of intensity 1



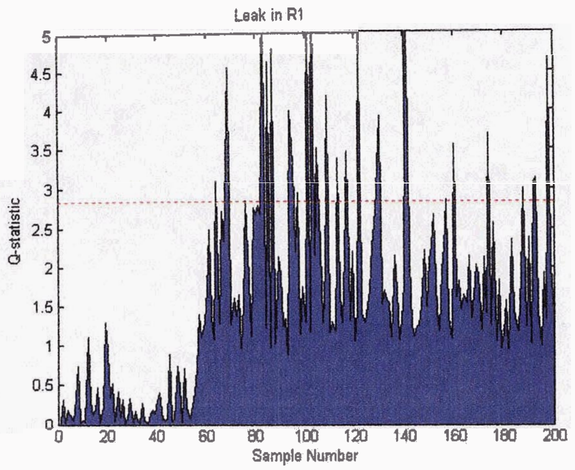


Figure A2.4 Q-statistic plots with 99% confidence limit for a single fault of intensity 1

BIBLIOGRAPHY

[1] X. Z. Wang, "Data Mining and Knowledge Discovery for Process Monitoring and Control," Springer, London, 1999.

[2] X. Z. Wang and R. F. Li, "Combining conceptual clustering and principal component analysis for state space based process monitoring," *Ind. Eng. Chem. Res.*, Vol. 38, pp. 4345-4358, 1999.

[3] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, The MIT Press, 1996.

[4] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Academic Press, 2001.

[5] X. Li, "Statistical Monitoring and Fault Diagnosing of Multivariate Processes Based on Nonlinear Principal Component Analysis," M.A.Sc. Thesis, Simon Fraser University, 2004.

[6] M. C. Johannesmeyer, "Abnormal Situation Analysis Using Pattern Recognition Techniques and Historical Data," M.Sc. Thesis, University Of California, Santa Barbara, 1999.

[7] J. E. Jackson, *A User's Guide To Principal Components*, John Wiley & Sons, Inc., 1991.

[8] V. Faber, J. G. Hochberg, P. M. Kelly, T. R. Thomas, and J. M. White, "Concept extraction – a data-mining technique," *Los Alamos Science*, Number 22, 1994.

[9] A. K. Conlin, E. B. Martin, and A. J. Morris, "Confidence limits for contribution plots," *Journal Of Chemometrics*, 2000, Vol. 14, pp. 725–736.

[10] R. L. Mason and J. C. Young, "Multivariate tools: principal component analysis," *Quality Progress,* February 2005, Vol. 38, No. 2, pp. 83-85.

[11] T. Kourti, "Process analysis and abnormal situation detection: from theory to practice," *IEEE Control Systems Magazine*, October 2002, Vol. 22, No. 5, pp. 10-25.

[12] E. B. Martin, A. J. Morris, and J. Zhang, "Process performance monitoring using multivariate statistical process control," *IEE Proc. of Control Theory and Applications*, March 1996, Vol. 143, No. 2.

[13] E. Cantú-Paz and C. Kamath, "On The Use Of Evolutionary Algorithms In Data Mining," Idea Group Publishing, 2001.

[14] M. J. Embrechts, B. Szymanski, and K. Sternickel, "Introduction to scientific data mining: Direct kernel methods and applications," Chapter 10 in *Computationally Intelligent Hybrid Systems*, Wiley Interscience, pp. 317-363, 2004.

[15] C. Fan, R. Guo, A. Chen, K. Hsu, and C.Wei, "Data mining and fault diagnosis based on wafer acceptance test data and in-line manufacturing data," *IEEE International Symposium. on Semiconductor Manufacturing*, October 2001, pp. 171–174.

[16] P. Yang and S. Liu, "Fault diagnosis for boilers in thermal power plant by data mining," *Control, Automation, Robotics and Vision Conference*, December 2004, Vol. 3, pp. 2176–2180.

[17] A. Singhal and D. E. Seborg, "Matching patterns from historical data using PCA and distance similarity factors," *Proceedings of the American Control Conference*, June 2001, Vol. 2, pp. 1759–1764.

[18] C. Zhang, L. Di, and Z. An, "Welding quality monitoring and management system based on data mining technology," *IEEE Proceedings of the Second International Conference on Machine Learning and Cybernetics*, November 2003, Vol. 1, pp. 13-17.

[19] C. Gertosio and A. Dussauchoy, "Knowledge discovery from industrial databases," *Journal of Intelligent Manufacturing*, February 2004, Vol. 15, No. 1, pp. 29-37.

[20] A. Singhal and D. E. Seborg, "Clustering multivariate time-series data," *Journal of Chemometrics*, 2005, Vol. 19, No. 8, pp. 427-438.

[21] M. C. Johannesmeyer, A. Singhal, and D. E. Seborg, "Pattern matching in historical data," *AIChE Journal*, September 2002, Vol. 48, No. 9, pp. 2022-2038.

[22] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis Part III: Process history based methods," *Computers and Chemical Engineering*, 2003, Vol. 27, pp. 327-346.

[23] P. R. Goulding, B. Lennox, D. J. Sandoz, K. J. Smith, and O. Marjanovic, "Fault detection in continuous process using multivariate statistical methods," *International Journal of Systems Science*, November 2000, Vol. 31, No. 11, pp. 1459–1471.

[24] B. M. Wise, N. L. Ricker, and D. J. Veltkamp, "Upset and sensor failure detection in multivariate processes," *AIChE Annual Meeting*, San Francisco, CA, November 1989.

[25] M. Kano, S. Hasebe, I. Hashimoto, and H. Ohno, "Fault detection and identification based on dissimilarity of process data," *Preprints of European Control Conference* (ECC), Porto, Portugal, September 2001, pp.1888-1893.

[26] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kauffman, 1993.

[27] J. R. Quinlan, http://www.rulequest.com, 2006.

[28] R. Kohavi and J. R. Quinlan, "Decision Tree Discovery," *Handbook of Data Mining and Knowledge Discovery*, Chapter 16.1.3, pages 267-276, Oxford University Press, 2002.

[29] X. Z. Wang and C. McGreavy, "Automatic classification for mining process operational data," *Industrial & Engineering Chemistry Research*, June 1998, Vol. 37, pp. 2215-2222.

[30] M. Chen, J. Han, and P. S. Yu, "Data Mining: An overview from a database perspective," *IEEE Transactions on Knowledge and Data Engineering*, December 1996, Vol. 8, No. 6, pp. 866-883.

[31] G. Dolanc, D. Juricic, A. Rakar, J. Petrovcic, and D. Vrancic, "Three-tank benchmark test", *Copernicus project*, CT94-02337, Jozef Stefan Institute, October 1996.

[32] A. Singhal and D.E. Seborg, "Pattern matching in historical batch data using PCA," *IEEE Control Systems Magazine*, October 2002, Vol. 22, No. 10, pp. 53-63.

[33] J. F. MacGregor and T. Kourti, "Statistical process control of multivariate processes," *Control Engineering Practice*, March 1995, Vol. 3, No. 3, pp. 403-414.

[34] G. M. Frey, "Multiresolutional Partial Least Squares and Principal Component Analysis of Fluidized Bed Drying," M.Sc. Thesis, University Of Saskatchewan, Saskatoon, Saskatchewan, March 2005.

[35] R. Singh, "A model to integrate Data Mining and On-line Analytical Processing: with application to Real Time Process Control," Ph.D. Thesis, Virginia Commonwealth University, Richmond, Virginia, June 1999.

[36] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, “A review of process fault detection and diagnosis Part I: Quantitative model-based methods,” *Computers and Chemical Engineering*, 2003, Vol. 27, pp. 293-311.

[37] S. J. Bailey, “From desktop to plant floor, a CRT is the control operators window on the process,” *Control Engineering,* 1984, Vol. 31, No. 6, pp. 86–90.

[38] P. Berkhin, “Survey of clustering data mining techniques,” Accrue Software, San Jose, CA, 2002.

[39] L. Berec and L. Tesaf, “Testing fault detection methods via three-tank system,” *Technical Report Issue A*, Copernicus Project CT94-0237, 1997.

[40] J. Lunze, “Laboratory three tanks system – benchmark for the reconfiguration problem,” *Technical report*, Tech. Univ. of Hamburg, Hamburg, Inst. of Control. Eng., Germany, 1998.