1-4-2016

Online Novelty Detection System: One-Class Classification of Systemic Operation

Ryan M. Bowen
rmb3518@rit.edu

R·I·T

Online Novelty Detection System: One-Class Classification of Systemic Operation

by

Ryan M. Bowen

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctorate of Philosophy in Microsystems Engineering

Microsystems Engineering Program
Kate Gleason College of Engineering

Rochester Institute of Technology
Rochester, New York
January 4, 2016

Online Novelty Detection System: One-Class Classification of Systemic Operation

by

Ryan M. Bowen

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctorate of Philosophy in Microsystems Engineering.

Dr. Ferat Sahin Professor, Electrical and Microelectronic Engineering	Date
Dr. Karl Hirschman Professor, Electrical and Microelectronic Engineering	Date
Dr. Sildomar Monteiro Assistant Professor, Electrical and Microelectronic Engineering	Date
Dr. Jason Kolodziej Associate Professor, Mechanical Engineering	Date
Aaron Radomski Chief Technologist, MKS ENI Products	Date

Certified by:

Dr. Bruce Smith Director, Microsystems Engineering Program	Date
Dr. Harvey J. Palmer Dean, Kate Gleason College of Engineering	Date

ABSTRACT

Kate Gleason College of Engineering
Rochester Institute of Technology

Degree: Doctorate of Philosophy **Program:** Microsystems Engineering
Author's Name: Ryan M. Bowen
Advisor's Name: Dr. Ferat Sahin
Dissertation Title: Online Novelty Detection System:
One-Class Classification of Systemic Operation

Presented is an Online Novelty Detection System (ONDS) that uses Gaussian Mixture Models (GMMs) and one-class classification techniques to identify novel information from multivariate times-series data. Multiple data preprocessing methods are explored and features vectors formed from frequency components obtained by the Fast Fourier Transform (FFT) and Welch's method of estimating Power Spectral Density (PSD). The number of features are reduced by using bandpower schemes and Principal Component Analysis (PCA). The Expectation Maximization (EM) algorithm is used to learn parameters for GMMs on feature vectors collected from only normal operational conditions. One-class classification is achieved by thresholding likelihood values relative to statistical limits. The ONDS is applied to two different applications from different application domains. The first application uses the ONDS to evaluate systemic health of Radio Frequency (RF) power generators. Four different models of RF power generators and over 400 unique units are tested, and the average robust true positive rate of 94.76% is achieved and the best specificity reported as 86.56%. The second application uses the ONDS to identify novel events from equine motion data and assess equine distress. The ONDS correctly identifies target behaviors as novel events with 97.5% accuracy. Algorithm implementation for both methods is evaluated within embedded systems and demonstrates execution times appropriate for online use.

To all my family, friends, and colleagues.

Acknowledgments

I would like to thank my advisor, Dr. Ferat Sahin, who over the past years has been a guide for many aspects of my life including, academic, career, and personal affairs. None of this work would have been possible without him, and I owe him a great debt. Dr. Sahin, thank-you so much and may our personal and professional relationship continue to grow stronger.

I would also like to thank:

- My parents, for their love and support for all my life. You have influenced me in so many ways.
- My fiancée, Alexandra Peruzzini for being so patient during the numerous long nights in the lab, and keeping me positive during times of adversity.
- My family, for supporting me during the long quest to obtain my Ph.D. it has been quite the journey but your support has kept me on track.
- The Multi-Agent BioRobotics Laboratory research assistants, for the wonderful lab atmosphere, hours of proof reading, and those late-night collective brain storming sessions.
- MKS ENI Products, Protequus LLC, and NRG Experimental LLC for funding my research and providing access to needed equipment.
- Aaron Radomski and Dan Sarosky, for their countless hours of data collection, productive status meetings, and for sharing some of their expert knowledge on RF power generators.
- Jeffrey and Michael Schab for their professional guidance and candid suggestions.
- My committee, for reviewing my work and suggesting improvements.
- Microsystems, Electrical and Microelectronic, and Computer Engineering Departments' faculty and staff for their assistance during my 12 years as an RIT student.

Contribution of the Research

- R.M. Bowen, F. Sahin, and J. Schab, "Early identification of equine distress: novel event detection of behavior," *Journal of Equine Veterinary Science*, 2015. Submitted for publication.
- R.M. Bowen, F. Sahin, and A. Radomski, "Systemic health evaluation of rf generators using gaussian mixture models," *Computers and Electrical Engineering*, 2015. Accepted for publication.
- R.M. Bowen, F. Sahin, A. Radomski, and D. Sarosky, "Embedded one-class classification on rf generator using mixture of gaussians," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, Oct 2015, pp. 2657-2662.

Contents

Abstract . . . . **iii**

List of Tables Captions . . . . **x**

List of Figures Captions . . . . **xiii**

1 Introduction **1**

1.1 Fault Detection and Semiconductor Industry 2

1.2 Health Monitoring 5

1.3 One-Class Classification 6

1.4 Novelty Detection Framework 10

1.4.1 Setup Phase - Feature Extraction and Selection . . 10

1.4.2 Initialization Phase - Dimensionality Reduction . . 11

1.4.3 Monitoring Phase - Detection of Faults 12

1.5 Learning Model Parameters 13

1.6 Online Novelty Detection System 15

2 Online Novelty Detection System **17**

2.1 Preprocessing 18

2.1.1 Translation 19

2.1.2 Normalization 20

2.2 Feature Vector Creation 21

2.3 Frequency Analysis 22

2.3.1 Bandpower 25

2.3.2 Feature Reduction - PCA 27

2.4 Mixture Models 29

2.4.1 Mixture Model Formulation 29

2.4.2 Gaussian Mixture Model 30

2.5 Maximum Likelihood Estimation 31

2.5.1 MLE Univariate Formulation 32

2.5.2 MLE Gaussian Estimators - Univariate 33
2.5.3 MLE Gaussian Estimators - Multivariate 34
2.6 Expectation-Maximization Algorithm 38
2.6.1 EM - Direct Approach 39
2.6.2 EM - Formulation as Incomplete-Data Problem . . 40
2.6.3 EM - Gaussian Mixture Models 42
2.7 One-Class Classification with GMM 44
2.7.1 Binary Decision Performance Metrics 46

3 RF Power Generator 49
3.1 System Variables 50
3.2 ONDS on RF Power Generator 53
3.2.1 Data Collection 54
3.2.2 Preprocessing 58
3.2.3 Feature Vector Creation 58
3.3 Experimental Results and Analysis 60
3.3.1 Validation of GMM and One-Class Classification . 60
3.3.2 ONDS Application to RF Power Generator 66
3.3.3 Final Parameter Evaluation 75
3.4 Embedded Implementation 78

4 Equine Distress 81
4.1 ONDS For Equine Distress 84
4.1.1 Data Collection and Preprocessing 84
4.1.2 Feature Vector Creation 86
4.2 Experimental Results 89
4.2.1 Experimental Setup 89
4.2.2 Novel Event Exploration Results 91
4.2.3 Individual Model Results 95
4.2.4 Generic Model Results 97
4.2.5 Embedded Implementation 99

5 Conclusion **104**

A Simplification and Derivations of MLE **107**

A.1 Simplification - Univariate Gaussian Distribution 107

A.2 Simplification - Multivariate Gaussian Distribution 108

A.3 Derivation - Multivariate Gaussian Distribution 109

References **112**

List of Tables

2.1 Quantities that may be derived from a confusion matrix. . . 47

3.1 MKS Regulus series RF power generator vital system variables. 52
3.2 MKS Regulus series RF power generator environmental system variables. 52
3.3 Subset of MKS Regulus series RF power generator lifetime (since build date) diagnostic indicators. 52
3.4 Dataset names, corresponding sample counts, and class descriptions for the standard datasets. 61
3.5 Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based on validation accuracy. 64
3.6 Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based solely on validation true positive rate. 64
3.7 Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based solely on validation specificity. 64
3.8 Number of mixtures and model threshold determined from accuracy grid search, but GMM is selected based solely on validation true positive rate. 65
3.9 GMM comparison with state-of-the-art classification methods. 66
3.10 RF power generator dataset information including variable, file, and unique unit counts. 67
3.11 Grid search method and parameter combinations explored. 68

3.12 Grid search results for each iteration, where the percent occurrence (%), average fitness (avg), and weight contribution (ω_c) for each method/parameter is reported. The reported "-" refer to a method/parameter that has been removed from the search in the previous iteration. 72
3.13 Grid search results without detrend operation where the percent occurrence (%), average fitness (avg), and weight contribution (ω_c) for each method/parameter is reported. . 74
3.14 Classification accuracy of normal and seeded-faults for RF power generator dataset using parameters chosen from results of iterative grid search experiments with the C5002 model. General processing methods are unit variance with PCA and a threshold of 3.0 standard deviations from mean training likelihood. Results are from 50 trial runs with 15 EM generations and model selection based on validation TPR. 76
3.15 Comparative results of k-means and GMM methods on classification of RF power generator data. Results are from 50 trial runs and 15 model generations per trial and selection based on TPR. 78
3.16 Embedded implementation's code footprint, memory usage and algorithm execution time. 80

4.1 Summary of test equine motion dataset. Each file contains 1024 data samples of 9-axis IMU data, windows are created using a moving window of size 1024 with window increment of 256, and total time is based on synchronized system clock times. 90
4.2 Summary of target behavior event count for all units. . . . 91

4.3 Novel event count per unit averaged over 50 trial runs with 25 EM each run using a model acceptance threshold scalar $\lambda = 3$ and mixture count $K = 3$. Standard deviations of numbers reported are provided in parenthesis. 96
4.4 Negative predictive values (NPV) per unit averaged over 50 trial runs with 25 EM each run and a model acceptance threshold scalar $\lambda = 3$ and mixture count $K = 3$. Standard deviations of numbers reported are provided in parenthesis. 96

List of Figures

2.1 High level system architecture of the Online Novelty Detection System. 18

3.1 Generalized block diagram for closed-loop system for an RF power generator with a subset of input/output signals generated during operation. 50
3.2 Offline Model Learning. 54
3.3 Online Classification. 55
3.4 Example raw fingerprints, showing three different variables including setpoint, power dissipation and PA current. . . . 56
3.5 Example fingerprint. 57
3.6 Average selection criteria weight with respect to mixture and band count for final grid search iteration. Algorithmic parameters fixed to use max-min local scaling, linear detrending, unit variance with PCA, and likelihood threshold scale of $\lambda = 3.0$. 73
3.7 Average selection criteria weight with respect to mixture and band count for final grid search iteration without using the detrend operation. Algorithmic parameters fixed to use max-min local scaling, unit variance with PCA, and likelihood threshold scale of $\lambda = 3.0$. 74

4.1 Example hand-segmented novel event of a roll behavior. . . 85
4.2 Single-sided power spectrum using FFT compared to the estimate of PSD using Welch's method. 87
4.3 Average novel window counts normalized to total window count per unit with respect to model acceptance values for different number of Gaussian mixtures. The merged consecutive novel windows are novel windows that are consecutively classified as novel and are counted only once. . . . 92

4.4 Average novel event count and negative predictive values with respect to model acceptance threshold values for a different number of Gaussian mixtures. 93
4.5 Average negative predictive values (NPV) for fall and rise behaviors for different model acceptance threshold values for different number of Gaussian mixtures. 94
4.6 Average time per novel event in seconds and total time normalized to the total collection time with respect to model acceptance threshold values for different number of Gaussian mixtures. 94
4.7 Generic model analysis presented as average values of 50 trials of negative predictive values with respect to mixture count and thresholding. For (a) each model's threshold value was set to be $\lambda = 3.0$ and for (b) the number of mixtures was set to $K = 4$. 98
4.8 Performance of generic behavior models versus individual behavior models with respect to NPV and novel event count. Values provided are average values across all units with 50 trials performed per unit. 99
4.9 Average execution time of 500 trials of novel window classification on different high-performance CPUs with Linux and Microsoft Windows operating systems. 101
4.10 Average execution time of 500 trials of novel window classification on the ARM® Cortex™-A8 with Linux operating system. 102

Chapter 1

Introduction

In recent years, there have been numerous methods proposed to achieve online health monitoring of various industrial systems. Most of these methods have not been specifically designed and/or executed on the targeted industrial system. Most industrial systems in the past have had limited processing power and memory available leading to a sparse number of solutions to embedded health monitoring systems. Conventional health monitoring uses rudimentary comparisons of current and nominal system characteristics. Gradual variation in nominal characteristics often cause conventional methods to fail and produce false indicators. Additionally, traditional methods fail when operational conditions produce system characteristics that fall outside nominal ranges solely on the basis that they were unanticipated/unobserved. These false indicators can cause system downtime, unnecessary diagnostic costs, and even material waste. Recently, increases in computational power and memory capacity for modern microprocessors has driven the feasibility of intelligent health monitoring systems within embedded environments. Therefore, a significant contribution to many application domains would be the realization of an embedded intelligent health monitoring system that is capable of online classification within an embedded environment. The semiconductor industry is one such application domain that can greatly benefit from such a monitoring system.

1.1 Fault Detection and Semiconductor Industry

The semiconductor industry's trend toward larger wafers (300mm to 450mm) for Integrated Circuit (IC) manufacturing demands reliable/available process equipment. To minimize the cost of ownership (COO), IC Process tools must not fail during IC fabrication processes. For example, RF plasma power sources are critical components used during etching and film deposition. Thus, the reliability/availability of these power sources is critical to maximize up-time and minimize cost of ownership. Furthermore, the ability to accurately determine a process tool's operational condition in vivo has huge potential for cost savings. Process monitoring and fault detection can help detect abnormal processes and equipment based on variation in process variables. There are four general process monitoring procedures: fault detection, fault identification, fault diagnosis, and process recovery [4]. Each of the four types of procedures are briefly defined in the following list.

Fault Detection - indicates that a fault has occurred.

Fault Identification - identifies the main effects (process variables) relevant to the fault.

Fault Diagnosis - determines which fault has occurred, location, time, etc. as well as the cause of the fault.

Process Recovery - removes the cause of the fault.

The focus of the proposed work is fault detection as an initial procedure in process monitoring and arguably the most important. Without accurate fault detection, identification and diagnosis cannot occur.

The bulk of current fault detection and classification of semiconductor manufacturing tools have used an Aluminum stack etch process as benchmark for their proposed methods [5, 6, 7, 8, 9]. Various machine learning techniques have been used in the detection of faults in semiconductor etch processes. Fault detection in processes have also been tested against benchmark simulation problems as in [10, 11]. Mahadevan et al. tested their one-class SVM fault detection and diagnosis on the Tennessee Eastman semiconductor etch process [10]. Park et al. applied their multi-class Support Vector Machine (SVM) to artificially generated fabrication data [11]. Hong et al. have used Modular Neural Networks (MNN) composed of Local Expert Networks [6]. Ison and Ison et al. have focused a dissertation on using a probabilistic model for fault classification of plasma equipment using predictions from tree-based methods and a Generalized Linear Model (GLM) for classification of different faults [12, 7]. Li et al. have used k-Nearest Neighbor (k-NN) with diffusion maps [8] and Yu et al. have used Gaussian Mixture Models (GMM) and bayesian inference [9]. Of these current machine learning methods, their successes is typically contingent on the availability of amount of data samples for training the classifiers.

With advances in metrology sensors, massive amounts of data are routinely collected such as temperature, pressure, flow rate, and power. The data obtained from these process variables can be monitored and control limits can be determined over time. Once established, these control limits can be analyzed using statistical process control (SPC) to detect if a process is within control. Current research has extended the use of SPC for fault classification. Goodlin et al. present a method that uses SPC charts to not only detect a fault but to also simultaneously classify the fault [5]. Typically, SPC may be used to identify a fault event but conventionally cannot classify the fault. Goodlin et. al's approach applies linear discriminant analysis to SPC charts to achieve simultaneous fault detection and classification. However, SPC in most cases is done offline and may cause a significant amount of scrap before control limits are established.

In recent years, the semiconductor industry has been focusing on the adaptation of advanced process control (APC) where one component of APC is fault detection [9]. These fault detection systems collect data from the manufacturing equipment sensors and attempt to quickly detect abnormal evolution of the process. Modern manufacturing equipment has many sensors and are able to produce a massive amount of data. This massive amount of data generally has many variables and causes univariate analysis such as SPC to be inadequate. Therefore, multivariate SPC (MSPC) have been developed and applied [13].

Many manufacturing processes have multiple steady-state conditions and operational states. This suggests that data from many manufacturing processes follow a multimodal model distribution and the standard application of MSPC may pose difficulties. To better model manufacturing processes, mixture models have been used. Using mixture models for fault detection poses a few issues; 1) how to model system characteristics that are not available during model initialization and 2) how to compensate for gradual changes.

The work that has been reviewed thus far has been with respect to fault detection for semiconductor manufacturing tools. There are other fields that have used fault detection as a component in a larger scope of system heath monitoring, such as health monitoring of equine animals [1].

1.2 Health Monitoring

There are a number online health monitoring systems that have been recently published that have used fault detection techniques. Of these monitoring systems it is seen that they span a range of application domains and methodologies. Ordaz-Moreno et al. present an automatic online diagnosis algorithm for detection of broken-rotor-bar in induction motors [14]. Ordaz-Moreno et al.'s utilize a mean-square function on a subset of Discrete Wavelet Transform (DWT) coefficients to determine health. Vanik et al. use a Bayesian framework for structural health monitoring [15]. Wang et al. provide an approach to anomoly detection in hard drives using the Mahalanobis distance [16]. Shakya et al. also use the Mahalanobis distance in combination with Gram-Schmidt Orthogonalization to health status for naturally progressing defects in bearings [17]. Many high-performance novelty detection algorithms are kernel based such as SVM, Regularization Networks (RN), Kernel Principal Component Analysis (KPCA), and Kernel Partial Least Squares (KPLS)[18]. Due to the time and memory complexities of these methods they are rarely used in an online manner. However, recently online formulations of Reproducing Kernel Hilbert Spaces (RKHS) [19] have lead to a number of publications with online kernel based model predictions [20, 21, 22, 23].

Many of the health monitoring systems reviewed utilize techniques that are highly dependent on the availability of information with respect to faulty conditions. It is possible to evaluate the health of a system by using information from a single condition assumed to be normal. The focus on learning only normal operational behavior has consequence when a system experiences operational behavior that has not yet been seen by the classifier. Thus, researchers are exploring ways to formulate classifiers where boundaries of the normal operation class are learned. Anything outside the learned boundaries is marked as non-normal or *novel*. These novel instances should be further analyzed by an expert to assess the physical operation of the system. The expert should then conclude the true operation of the system as normal, non-normal, or possibly faulty. One-class classification is a type of classification technique that is used often in novelty detection.

1.3 One-Class Classification

In machine learning, the basic assumption is to have a training dataset which represents some parameter/feature space of the system, such that all the possible inputs and outputs are independent and identically distributed (i.i.d) random variables. When there is no practical way of obtaining faulty data from all possible fault patterns, it is common that there is significantly more information for normal operation than faulty operation. In most of the fault analysis work, a multi-class classifiers are used. In the multi-class cases, the machine learning approaches such as Support Vector Machines, Neural Networks, Radial Basis Function Networks, and Bayesian Networks are efficiently used; given there are enough samples from the non-normal operation class. Machine learning approaches depend on the data available for training. Henceforth, the classifiers can be only as successful as the samples used during training, and for multi-class classifiers, both normal and non-normal operational data must be available. However, there is a common difficulty in obtaining data for non-normal operation for many systems. Researchers realize the challenges of obtaining non-normal data, and suggest a paradigm shift to learn only the normal operation of a system rather than attempting to differentiate non-normal operation from normal.

The one-class classification term originates from Moya et al.'s [24] classification work with neural networks and their application to target recognition in synthetic aperture radar. One-class classification's primary assumption is that information is only available for *one* class, the *target* class. The problem of one-class classification is to define a boundary around the target class as to maximize object acceptance while minimizing outlier acceptance [25].

Two of the most popular categorical methods to realizing a one-class classifier are density-based and boundary-based. For density-based methods an estimate of the density of training data is found and a threshold value is set. To estimate density many different distributions may be used including Gaussian, Poisson, and mixture of Gaussians. For training data with sufficiently high sample sizes and proper model assumption, density-based one-class classification can be very effective. In the case of limited samples in the training data, it may not be appropriate to generalize the problem as done in density-based methods. Therefore, a better approach is to solve the problem available and use boundary-based methods to define the target data. Some of the most common boundary-based methods used are k-means, k nearest neighbors (k-NN), and Support Vector Data Description (SVDD). It is noted that boundary-based methods are able to work on small sample sizes but are highly dependent on distance measures and thus are sensitive to feature scaling [25].

A review of current one-class classification approaches has yielded a few prominent terms in relation to one-class classification; *anomaly detection*, *outlier detection*, *novelty detection*, and *concept learning*. Some of these terms are very similar in definition but deviate with respect to their application to one-class classification. Anomaly and outlier detection are used synonymous to novelty detection and concept learning but have slightly different characteristics. A few definitions/descriptions from literature have been selected for these terms to stress their subtle differences and to avoid improper use.

- ANOMALY/OUTLIER DETECTION - The most broad method in approach to one-class classification where the problem is to find patterns in data that do not conform to expected behavior. These nonconforming patterns may indicate noise, deviations or exceptions [26].

- NOVELTY DETECTION - Smaller subset of anomaly/outlier detection where the problem is to identify unobserved (*novel*) data or signal that a machine learning system is not aware of during training [27]. Additionally, novelty detection will typically incorporate the novel information into normal model after detection [26].

- CONCEPT-LEARNING - Typically focuses on discriminate-based learning that relies on both examples and counter examples of the *concept*. Concept learning may be applied to one-class classification through recognition-based learning systems that do not require the use of counter-examples [28].

Despite the closely related definitions and uses of the above terms, *novelty detection* has been chosen as the appropriate focus as a strategy for the proposed work. The detection of *non-normal* system operation, is not necessarily considered noise nor as exception. Additionally, complex system adaptation may yield deviations that are exclusively an effect of unobserved system normality. Therefore, novelty detection has been chosen specifically for its capacity to detect and potential to incorporate previously unobserved information.

Novelty detection spans many application domains including Internet Telecommunication (IT) security, healthcare informatics, medical diagnostics and monitoring, industrial monitoring, text mining, and sensor networks. There have been a number of surveys on novelty detection that have reviewed the various theoretical approaches and various categorical techniques. In 2001, Tax's Ph.D. dissertation classified novelty detection, as it pertains to one-class classification, into three approaches: density-based, boundary-based and reconstruction-based [25]. A review conducted by Markou and Singh focuses on statistics-based and neural network-based approaches [27, 29]. A review by Pimentel et al. provides a more recent and extensive review on novelty detection [30], where they highlight novelty detection across five application domains; probabilistic, distance-based, domain-based, reconstruction-based, and information theoretic. Ding et al. provides a recent experimental evaluation of current novelty detection methods including SVDD [31], k-means, k nearest neighbors (k-NN), and Gaussian mixture method (GM) [32].

Novelty detection using one-class classification is a general description of an abstract process. The actual implementation of the one-class classification may be done using a variety of techniques. However, a general approach to the problem has been proposed by Filev et al. [33], where a Novelty Detection Framework (NDF) has been described as an application to online health monitoring of bearings.

1.4 Novelty Detection Framework

The Novelty Detection Framework is capable of updating a decision model continually and autonomously using unsupervised learning methods. The outcome of the NDF is a generic and effective monitoring system capable of detecting novel and/or abnormal equipment conditions prior to the actual event. NDF has been experimentally tested as an application to bearing monitoring with none to very few false alarms. The NDF is discussed in more detail as it will serve as the primary basis of the proposed work and will be used for comparison during experimental analysis. The NDF is composed of three phases 1) Setup 2) Initialization and 3) Monitoring. The following subsections discuss the general process of the different phases of NDF. Full explanation and implementation details of the NDF span multiple publications. Thus, the exact specifics such as equations and algorithmic descriptions are out of the scope of this document but may be found in further detail in [33], [34, 35, 36].

1.4.1 Setup Phase - Feature Extraction and Selection

The setup phase of the NDF is to transform the raw signal of the system into features. The type of features whether it be time domain, frequency domain or a mix is selected based on the application. The end result from the setup phase is that some K dimensional feature vector is generated. The setup phase of the NDF is the only non-generic aspect of the framework and requires some prior knowledge of the dynamics of the system in order to make proper assumptions in the transformation process.

1.4.2 Initialization Phase - Dimensionality Reduction

During the initialization phase a predetermined number of feature vectors (K) are collected. After the collection of the K feature vectors, the initialization data set has dimensions ($N \times K$) and in most cases N can be very large. High dimensionality poses significant computational challenges therefore the first process of the initialization phase of NDF is dimensionality reduction. By default NDF uses Principal Component Analysis (PCA) to realize dimensionality reduction. NDF as implemented in [33] only uses the first two principal components (PCs) resulting in a two-dimensional (2D) PC space. The 2D PC space reduction was decided as to allow for visualization of the space. The 2D space visualization is used as a simplified output such that a non-expert of the system could quickly evaulate clusters within the feature vector space. Once the PCA matrix is computed to transform the data set from ($N \times K$) to ($N \times 2$), the data is clustered to determine the number of Operational Modes (OMs) that are present in the initialization data. NDF uses the Greedy Expectation Maximization (EM) Clustering Algorithm [36] to identify the initial number clusters, their centers and covariances. These centers and covaraince may be used as parameters to various distributions that in turn can define cluster boundaries.

1.4.3 Monitoring Phase - Detection of Faults

The NDF is capable of real-time prediction of two different faults: Incipient and drastic faults. Incipient faults are associated with small gradual changes typically related to wear and drift of system characteristics. Drastic faults are consequent of sudden changes in system dynamics and often quickly lead to system failure. The detection of faults is executed during the monitoring phase of the NDF and may be done either online or offline.

Prior to fault detection, preprocessing and cluster updates are performed. During preprocessing, the current feature vector is transformed into the 2D PC space using an updated PC matrix. The updated PC matrix is calculated by performing SVD using weighted combinations of the previous and current mean and covariances of the feature vectors. The weighting is controlled through a learning parameter α which may be used to quantify the influence of new data on model parameters. After transforming the new feature vector, the clusters are updated using a modified version of the k-nearest neighbor (k-NN) rule [34].

The ability of the NDF to detect incipient faults is based on its ability to predict the OM clusters dynamics using the evolving Takagi-Sugeno (eTS) model [35]. The eTS models provide predicted values of the elements of transformed feature vectors (TFVs). From the predicted TFVs' values, a trajectory of the TFVs are calculated and used to determine time predictions of when a TFV will cross their OM cluster boundary. These predictions are used to determine which OM cluster a TFV will belong. A prediction of incipient failure is quantified by NDF using a TFV's predicted OM cluster and its health factor. The health factor is a measure based on the age of the OM cluster and the TFVs belonging to the cluster.

The detection of drastic faults by the NDF is linked to the rapid creation of new OM clusters during monitoring. The assumption that is made with NDF is that drastic faults are an indication of dramatic and abrupt changes in system dynamic. These abrupt changes are assumed to be linked to the creation of new OM clusters that have limited sets of feature vectors. To track these significant changes in OM clusters, an Exponential Weight Moving Average (EWMA) SPC chart is used. The EWMA chart tracks mean and variance of OM cluster count and a drastic fault is identified when the EWMA chart is determined to be out of control.

The NDF is a general framework that can be used for online health monitoring of a system. It has been designed such that many of the components of the framework can be replaced by other techniques. A vital component of the NDF is the clusters that are learned using the EM algorithm. Instead of using clusters, an extension would be to use mixture models. With mixture models, a challenge is the process of accurately and efficiently learning the model parameters.

1.5 Learning Model Parameters

The standard approach to learning parameters to mixture models is the EM algorithm. However, some of the well known caveats of the EM algorithm is its sensitivity to initialization and assumption that the number of components within the mixture are known.

Numerous model selection criteria have been developed to better choose the number of components for a mixture model and have been extensively reviewed in [37]. Of the model selection criteria, some of the most commonly used and well established are the Minimum Description Length (MDL) [38], Akaike Information Criterion (AIC)[39], Bayesian Information Criterion (BCI) [40], and Likelihood Ratio Test (LRT) [41]. Many extensions to EM have been presented in literature in addition to and in lieu of well established model selection criterion. The most common extension to EM is to incorporate greedy search techniques as a method to determine the best number of components to use in a mixture model. Verbeek et al. and Vlassis et al. have provided such greedy methods to EM [36, 42].

Other approaches to improving the EM algorithm have been to integrate other heuristic search algorithms such as Genetic Algorithms (GA), Particle swarm optimization, and various others. Genetic approaches have been the most prevalent heuristic method due to their ease of integration into the existing EM algorithm. Pernkopf et al. have used an elitist GA with MDL as the fitness function to find parameters to Gaussian mixture models [43]. Pernkopf et al.'s GA-EM is two-fold as it first performs a set number of EM iterations on each member in the GA population then recombination, selection, and mutation is performed on the EM updated members. Pernkopf et al.'s GA-EM has shown improvement over EM on artificial datasets, but is susceptible to outliers. Some other notable GA applications to EM may be found in [44, 45, 46, 47].

Particle swarm optimization has been used to help improve global search for EM. Guan et al. have used a Discrete PSO (DPSO) to solve for parameter estimates in an alternating fashion with EM [48]. The alternating DPSO and EM approach help global search but also avoids lengthy convergence times that are common for PSO. Particle swarm optimization has also been used with EM to solve difficult partials or integrals that may arise based on model selection. Zhang et al. have used a PSO-EM where PSO was applied to the one of the steps of EM as an assist to computing integrals of a normal compositional model [49]. Furthermore, PSO has been used as a replacement for EM as a density estimator. Yan and Osadciw have developed a Dimension Adaptive PSO (DA-PSO) that does not require the calculation of parameters from likelihood but instead uses the mixture model's PDF directly in a fitness function [50]. The DA-PSO has advantage over EM and other PSO methods as it does not require knowledge of dimensionality.

Review of current machine learning techniques and their applications to fault detection in various applications has outlined the demand for such methods. Furthermore, the current status of health evaluation of systems leave potential for improvement through online classification methods capable of execution within an embedded system. Thus, leading to the proposed work of an online novel detection system.

1.6 Online Novelty Detection System

The proposed Online Novelty Detection System (ONDS) uses Gaussian Mixture Models (GMMs) and one-class classification techniques to model normal systemic operation and identify non-normal or novel operation. To test the realization of the ONDS, two different problems from different application domains are considered. The first application of the ONDS is to identify novel systemic operation of RF power generators and the second is to detect novel events in equine animal behavior. Both applications assume patterns in time-series data provided by sensors. However, the nature of the variables themselves differ slightly for each application. The data collection procedure for the RF power generator application is performed by a controlled procedure and the signals themselves are less random, whereas the equine motion data is naturally random. These two different system signals allow actual testing of the ONDS as a generic approach and to identify methods that require per application modifications.

In terms of originality, this work provides an extensive analysis of how processing methods applied to raw data affect modeling and classification parameter selection. In general, research with one class classifiers emphasizes optimization of the classifier itself. Instead, this work presents the one-class classifier as a generic solution and focus is placed on parameter selection. Additionally, the proposed system is applied to two different signals, the first being periodic and the second random. Machine learning classification techniques such as the one presented in this work are rarely implemented in an actual embedded system. Therefore, with respect to algorithmic contribution, the work presented realizes the entire algorithm within an actual embedded system for real-time use.

The remainder of this work follows by generalizing the proposed method of an Online Novelty Detection System in Chapter 2. Along with the generalization of the ONDS, any significant theory or background is provided as it is mentioned. Chapters 3 and 4 are specific applications of the ONDS. Chapter 3 covers the use of ONDS for novelty detection in RF power generators for fault detection. Chapter 4 is novelty detection of equine behavior for detection of distress. Conclusion of the ONDS and its results from the applications are summarized in Chapter 5.

Chapter 2

Online Novelty Detection System

The proposed work is an Online Novelty Detection System (ONDS) that uses Gaussian Mixture Models (GMMs) and one-class classification techniques as a general approach to assessing systemic health. The ONDS is designed to be embedded and executed within an embedded system to provide online health evaluation. From Figure 2.1, the execution of the proposed ONDS is also paired with an offline model learning stage.

Offline Model Learning Stage: During the offline model learning stage, parameters are learned that estimate a model that represents *normal*. The model is estimated using a set of features generated from data collected during known *normal* operation. Additionally, the offline model learning stage learns, calculates, and/or defines other required algorithmic parameters.

Online Classification Stage: This stage uses one-class classification techniques to identify whether current systemic operation belongs to what has been learned to be *normal*. The online classification stage uses similar procedures as the offline stage. The major difference of the online stage to the offline stage is that during the online stage the model and other algorithmic parameters are known, since they were learned, set

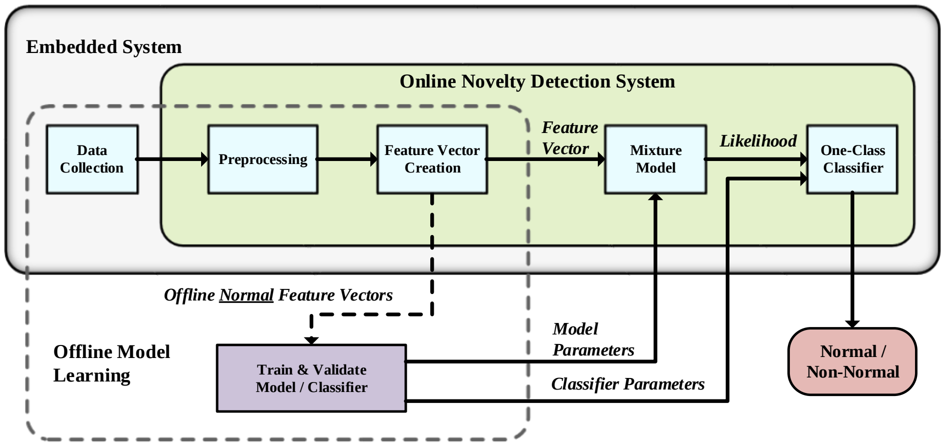


Figure 2.1: High level system architecture of the Online Novelty Detection System.

and/or calculated during the offline stage. The overall functional blocks are explained by the following subsections: *Preprocessing*, *Feature Vector Creation*, *Mixture Models*, and *One-class Classification*. Data collection is not included here as it is specific to each application.

2.1 Preprocessing

Data provided by a system may not be in a form best suited for statistical analysis or other analytical methods. Thus, various preprocessing methods may be used to scale, translate, and/or transform the original data into a form more suitable for particular analytical methods. Following are some of the preprocessing methods that are used in this work. Each of the preprocessing methods are presented explaining why they are used. The applicability of the methods is covered in Chapters 3 and 4 through empirical data.

Statistical information such as the mean, maximum, minimum, and variance are useful during preprocessing and their values are represented in vector form as ($\mathbf{Mean}_{[1\times M]}$, $\mathbf{Max}_{[1\times M]}$, $\mathbf{Min}_{[1\times M]}$, and $\mathbf{Var}_{[1\times M]}$; where M is the number of variables in the data. Often preprocessing is performed on multiple data samples, thus for computational efficiency and notation simplicity, matrix-based formulations are used. The vectors are converted to matrix forms through replication, as represented by Equation (2.1), where $\mathbf{A}_{[1\times M]}$ is one of the mean, maximum, minimum, or variance vectors and $\mathbf{A}_{[N\times M]}$ is the resulting matrix representation with N number of samples.

$$\mathbf{A}_{[\mathbf{N}\times\mathbf{M}]} = \begin{pmatrix} \mathbf{A}_{[1\times M]} \\ \mathbf{A}_{[1\times M]} \\ \vdots \\ \mathbf{A}_{[1\times M]} \end{pmatrix} \tag{2.1}$$

2.1.1 Translation

Data translation is useful to remove bias in the data and/or unwanted trends. Many analytical methods translate data such that the individual variables have no bias or their mean is zero (*zero-mean*). Zero-mean data helps reduce large 0 Hz (DC) bias that can occur after transforming data into the frequency domain. The mean used for translating the data may be chosen to be some global mean of the variables or may also be the local (current sample set) variable's mean. Equation (2.2), represents the zero-mean method which is applied column-wise, where $\mathbf{Y}$ is a data sample with N observations and M variables ($[N \times M]$). The $\mathbf{Mean}_{[N\times M]}$ represents the column-wise mean of $\mathbf{Y}$ repeated for N rows such that matrix subtraction can be done efficiently/correctly.

$$\tilde{\mathbf{Y}} = \mathbf{Y} - \mathbf{Mean}_{[\mathbf{N}\times\mathbf{M}]} \tag{2.2}$$

Some data series have trends that may distort relationships thus detrending operations can help remove these trends. A simple method to detrending is to compute the linear least-squares fit for each variable in the series and use the linear estimator to subtract the trend. To detrend data sample $\mathbf{Y}$ with N observations and M variables, the domain for each variable is fixed to be $\mathbf{x} = (1, 2, \ldots, N)^T$. The first order coefficients of the linear least-squares fit, $\hat{\boldsymbol{\beta}}$, may be found using Equation (2.3). Then using $\hat{\boldsymbol{\beta}}$, the detrended data sample may be estimated as $\hat{\mathbf{Y}}$ by using Equation (2.4). Following are some normalization methods commonly used during preprocessing.

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\mathbf{Y} \tag{2.3}$$

$$\hat{\mathbf{Y}} = \mathbf{Y} - \mathbf{x}\hat{\boldsymbol{\beta}} \tag{2.4}$$

2.1.2 Normalization

Data normalization affects the range of the variables and is performed because in many cases variables do not have the same ranges. With respect to classification and classifiers based on distance measures, these disproportionate ranges can cause some variable to dominate distance measures. Three normalization techniques that can help reduce this effect are *max-min*, *max*, and the *z-score*. The *max-min* normalization method, from Equation (2.5), re-scales a variable to the range $[0, 1]$, where // is element-wise matrix division.

$$\tilde{\mathbf{Y}} = \left(\mathbf{Y} - \mathbf{Min}_{[N\times M]}\right) // \left(\mathbf{Max}_{[N\times M]} - \mathbf{Min}_{[N\times M]}\right) \tag{2.5}$$

The *max* scaling method, from Equation (2.6), re-scales the variables such that their maximum value is 1 and their minimum is based on their original ranges. The *max* scaling method is particularly attractive if the variables have original minimum values of zero.

$$\tilde{\mathbf{Y}} = \mathbf{Y} // \left(\mathbf{Max}_{[N\times M]}\right) \tag{2.6}$$

Another normalization method is the *z-score* or standard score as calculated by Equation (2.7). The z-score scales the variable to have unit variance and to also have zero-mean.

$$\tilde{\mathbf{Y}} = \left(\mathbf{Y} - \mathbf{Mean}_{[N\times M]}\right) // \left(\mathbf{Var}_{[N\times M]}\right)^{\frac{1}{2}} \tag{2.7}$$

After data has been preprocessed, features are extracted to generate feature vectors that will be used during model learning.

2.2 Feature Vector Creation

The process of feature vector creation is typically driven by the application and therein its data. The application(s) of focus in this work are those that yield time-series data. From the time-series data, a system under normal operation is assumed to have some periodic pattern(s) that differ from non-normal operation. Therefore, frequency components of the time-series data are chosen as the *features* to extract and used to create feature vectors. Using frequency analysis techniques, the preprocessed time-series data can be transformed into the frequency domain and frequency features can be extracted.

2.3 Frequency Analysis

The direct approach to obtaining frequency information from a signal is to use the Fourier transform. However, the signals provided are almost always discrete samples, therefore the Discrete Fourier Transform (DFT) is used. The DFT produces a finite ordered list of complex coefficients that describe a combination of complex sinusoids [51]. These coefficients can be found by using Equation (2.8), and may be used as the features to be modeled. Where $x\left[n\right]$ is a discrete signal with N samples.

$$F[n] = \sum_{n=1}^{N-1} x[n] e^{-j2\pi\left(\frac{kn}{N}\right)} \tag{2.8}$$

Given a two-dimensional time-series data sample, the 2D-DFT may also be used by performing the DFT column-wise and then row-wise on its result. These two DFT operations may be merged using by using Equation (2.9). Where $x\left[n, m\right]$ is a discrete with M signals and N samples.

$$F\left[u, v\right] = \frac{1}{MN} \sum_{n=1}^{N-1} \sum_{m=1}^{M-1} x[n, m] e^{-j2\pi\left(\frac{u}{N}n + \frac{v}{M}m\right)} \tag{2.9}$$

Generally, the DFT is implemented using the Fast Fourier Transform (FFT). The FFT is an algorithm that is capable of reducing the complexity of computing the DFT from $O\left(N^2\right)$ to $O\left(N \log N\right)$. The Cooley-Tukey implementation of the FFT [52] is the most common and may be acheived using a recursive function as listed in Algorithm 1. Similar to the 2D-DFT the 2D-FFT can be achieved by performing two FFT operations. The power of the frequency components (power spectrum) is obtained by calculating the absolute value of the complex DFT/FFT coefficients. However, with a signal that is more random and less periodic, a more accurate method to describe the frequency spectrum is to estimate the power spectral density (PSD).

Algorithm 1 Cooley-Tukey Recursive FFT Algorithm

```
function FFT(&$x$)                                ▷ reference to complex array
    $N \leftarrow \text{length}(x)$
    if $N \leq 1$ then
        return 1
    end if
                                                  ▷ Divide into even/odd
    $even \leftarrow x[0, 2, \ldots, N-1]$
    $odd \leftarrow x[1, 3, \ldots, N-1]$
                                                  ▷ Conquer
    FFT(even)
    FFT(odd)
                                                  ▷ Combine Results
    for $k = 0$ to $N/2$ do
        $t \leftarrow exp(-2\pi k/N) * odd[k]$
        $x[k] \leftarrow even[k] + t$
        $x[k + N/2] \leftarrow even[k] - t$
    end for
end function
```

The PSD estimate can be determined using Welch's method of periodogram averaging [53]. Welch's method sections the window of data, taking the periodogram of the sections and calculating the average of the sectioned periodograms. Prior to application of Welch's method a Hamming window is applied to reduce the effect of spectral leakage. The Hamming window values are calculated using Equation (2.10) where T is the number of samples desired for each segment [51].

$$w\left(n\right) = 0.54 - 0.46\cos\left(2\pi\frac{n}{T-1}\right),\ 0 \leq n \leq (T-1) \tag{2.10}$$

The implementation of Welch's function uses the FFT to calculate the sectioned periodograms, with an overlap of 50% and at most eight sections. The actual number of sections may be determined by using Equation (2.11), where N is the size of original window and N_O is the number of overlap samples as calculated by Equation (2.12).

$$S = \left\lfloor \frac{N - N_O}{T - N_O} \right\rfloor \tag{2.11}$$

$$N_O = \left\lfloor \%\text{overlap} * T \right\rfloor \tag{2.12}$$

The FFT used to calculate the periodograms is the *N-point* FFT where the number of samples are zero-padded to equal N_{FFT} as calculated by Equation (2.13).

$$N_{FFT} = 2^{\lceil \log_2(N_O) \rceil} \tag{2.13}$$

Regardless of using the PSD or FFT directly, both provide a double-sided spectrum. Most applications are symmetrical about 0 Hz or DC and only the positive half of the spectrum is required. Therefore, the number of coefficients, N_C, is expressed by Equation

(2.14) where N_{FFT} are the number coefficients used when performing the FFT.

$$N_C = \frac{N_{FFT}}{2} + 1 \tag{2.14}$$

When considering random signals with slight periodicity, the frequency components are not deterministic and may reside in a range of frequencies or a frequency band. The average power in a band of frequencies or bandpower may be used as features instead of the direct FFT/DFT/PSD power coefficients.

2.3.1 Bandpower

Banding adjacent power coefficients, in this case, relates to finding the power within frequency ranges or *bands*. If specific frequency bands are unknown then the number of elements in each band is dictated by the desired number of bands and the number of FFT power coefficients (N_C). Our approach to calculating the bandpowers attempts to evenly distribute the number of elements in each band. Uneven division is handled by using slightly more elements within the lower frequency ranges. More elements per band for the lower frequencies was chosen to allow for better separability of the high frequency components. The number of elements in each band is calculated as:

$$\Delta = \left\lfloor \frac{N_C}{\beta} \right\rfloor \tag{2.15}$$

where N_C is the number of FFT coefficients and β is the desired number of bands. The number of remaining elements due to uneven band distribution is:

$$r = N_C - \Delta\beta \tag{2.16}$$

After determining Δ and r, the actual bandpower per band is accumulated as:

$$\mathbf{y}_k^B[t]=\begin{cases}\sum_{j=1}^{\Delta+1}\mathbf{y}_k^P[(t-1)(\Delta+1)+j], & \text{if } t<=r\\ \sum_{j=1}^{\Delta}\mathbf{y}_k^P[r(\Delta+1)+(t-r-1)\Delta+j], & \text{otherwise}\end{cases} \tag{2.17}$$

where $\mathbf{y}_k^P$ is the matrix of power values for the power spectrum coefficients for the k^{th} variable, $k = 1, \ldots, M$, $t = 1, \ldots, \beta$, Δ is the general number of elements in each band, and r is the number of low frequency band containing an extra element. After computing the bandpowers, $\mathbf{Y}^B$ is reshaped into a single feature vector, because the overall goal is to classify the signal as a whole. Thus, a single feature vector is a single row-wise vector of size $1 \times \beta \cdot M$, where M is the number of variables in the signal and β is the number of bands used in the bandpower computations. Even after computing the bandpower, the number of features can still be large and many of them potentially insignificant. Therefore, additional feature reduction is performed using Principal Component Analysis (PCA).

2.3.2 Feature Reduction - PCA

Principal Component Analysis (PCA) is a statistical procedure that transforms a set of possibly related variables into a set of linearly uncorrelated variables that are called principal components (PCs) [54]. These PCs refer to orthonormal eigenvectors that may be calculated using matrix factorization techniques. These eigenvectors have eigenvalues that correspond to the amount of variance that exists in the data in the direction of their respective eigenvectors. With respect to feature vectors, PCA is specifically used to find a lower dimensional encoding for a feature vector by using an orthogonal projection of the feature vector onto the column space spanned by eigenvectors, as shown in Equation (2.18)[55].

$$\hat{\mathbf{y}}_j = \mathbf{y}_j \mathbf{W} \tag{2.18}$$

The optimal solution may be obtained by Equation (2.19), where $\mathbf{V}_L$ are the L eigenvectors with largest eigenvalues of the covariance matrix.

$$\hat{\mathbf{W}} = \mathbf{V}_L \tag{2.19}$$

To define a solution to PCA in terms of eigenvectors, the Singular Value Decomposition (SVD) matrix factorization method is used. Using SVD, $\mathbf{Y}$ can decomposed by using Equation (2.20), where the columns of $\mathbf{U}$ are the left singular vectors, columns of $\mathbf{V}$ are the right singular vectors, and $\mathbf{S}$ is a matrix with the main diagonal containing the singular values and is 0 elsewhere. With respect to feature reduction, $\mathbf{Y}$ is the set of feature vectors that are under analysis and used to determine the PCs.

$$\mathbf{Y} = \mathbf{USV}^T \tag{2.20}$$

Using SVD a connection between eigenvectors and singular vectors can be established, as represented in Equation (2.21) and (2.22), where $\mathbf{Y} = \mathbf{USV}^T$ and $\mathbf{D} = \mathbf{S}^2$.

$$\mathbf{Y}^T\mathbf{Y} = \mathbf{VS}^T\mathbf{U}^T\mathbf{USV}^T = \mathbf{V}\left(\mathbf{S}^T\mathbf{S}\right)\mathbf{V} = \mathbf{VDV}^T \tag{2.21}$$

$$\left(\mathbf{Y}^T\mathbf{Y}\right)\mathbf{V} = \mathbf{VD} \tag{2.22}$$

The eigenvectors of $\mathbf{Y}^T\mathbf{Y}$ are equivalent to $\mathbf{V}$ which are the right singular vectors of $\mathbf{Y}$. Additionally, $\mathbf{D}$ represents the eigenvalues of $\mathbf{Y}^T\mathbf{Y}$ and equals to the squared singular values. Referring to Equation (2.19), $\hat{\mathbf{W}} = \mathbf{V}_L$ and the eigenvectors $\mathbf{V}_L$ are selected from a truncated SVD using a rank L approximation. The value L is minimum number of eigenvalues whose sum is $p\%$ of the total sum of all eigenvalues as shown in Equation (2.23).

$$L = \underset{l}{\operatorname{argmax}} \left\{ l \;\middle|\; \sum_{i=1}^{l} D\left(i,i\right) \leq tr\left(D\right) \cdot p \right\} \tag{2.23}$$

After learning the PCA matrix $\hat{\mathbf{W}}$, feature vector(s) are projected into the PC space. These reduced feature vector(s) are used during mixture model learning and one-class classification. Next the formulation of mixture models and learning model parameters are discussed.

2.4 Mixture Models

Finite mixtures of distributions are commonly used as a mathematical method to statistical modeling. Mixture models have been applied to many fields and are used as techniques to clustering and latent class analysis. More generally mixture models are attractive due to their ability to provide descriptive models for data distributions. A traditional mixture model with respect to clustering assumes that a set of N, M-dimensional data vectors $\mathbf{y}_1, \ldots, \mathbf{y}_K$ can be represented as a finite number of K components with some unknown proportions $\pi_1, \ldots, \pi_N$ [37]. To understand mixture models parameters and obtain their estimates, a formal definition of mixture models is provided.

2.4.1 Mixture Model Formulation

The general formulation of a mixture model follows similar notation as presented by McLachlan and Peel [37]. Let $\mathbf{y}_1, \ldots, \mathbf{y}_N$ denote an observed random sample such that $\mathbf{y}_j$ is a random vector with probability density function $f(\mathbf{y}_j) \in \mathbb{R}^M$ and $j = 1, \ldots, N$. Therefore, $\mathbf{y}_j$ contains M measurements of the jth observed sample of random variables and $\mathbf{Y} = \left(\mathbf{y}_1^T, \ldots, \mathbf{y}_N^T\right)^T$ contains all the observed random samples. Therefore, the probability density function of an observed random vector $\mathbf{y}_j$, in parametric form, for a K-component mixture model is formulated by Equation (2.24).

$$f(\mathbf{y}_j; \mathbf{\Psi}) = \sum_{i=1}^{K} \pi_i f_i(\mathbf{y}_j; \boldsymbol{\theta}_i) \tag{2.24}$$

Here $\boldsymbol{\Psi}$ contains all the unknown parameters belonging to the mixture model and is written according to Equation (2.25), where $\boldsymbol{\xi}$ contains all the *a priori* parameters in $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$.

$$\boldsymbol{\Psi} = \left(\pi_1, \ldots \pi_K, \boldsymbol{\xi}^T\right)^T \tag{2.25}$$

Furthermore, let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)^T$ be the vector of mixing proportions, where $\sum_{i=1}^{K} \pi_i = 1$. Equation (2.24) is the general form of a mixture model. Actual use requires an assumption of a distribution, where the most frequently used is the Gaussian distribution.

2.4.2 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a mixture model where the data is assumed to approximate a Gaussian distribution or likewise is assumed to be normally distributed. The probability density function of an observed random vector, $\mathbf{y}_j$, as a Gaussian mixture model of K normal components is formalized by Equation (2.26), where $i = \{1, \ldots, K\}$ and $\sum_{i=1}^{K} \pi_i = 1$.

$$f_{\mathcal{N}}\left(\mathbf{y}_j; \boldsymbol{\Psi}\right) = \sum_{i=1}^{K} \pi_i \phi\left(\mathbf{y}_j; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) \tag{2.26}$$

Furthermore, the generalized multivariate Gaussian density function, ϕ, is given by Equation (2.27), where $|\cdot|$ is the determinant operator, M is the dimension cardinality of $\mathbf{y}_j$.

$$\phi\left(\mathbf{y}_j; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) = (2\pi)^{-\frac{M}{2}} \left|\boldsymbol{\Sigma}_i\right|^{-\frac{1}{2}} exp\left(-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1}\left(\mathbf{y}_j - \boldsymbol{\mu}_i\right)\right) \tag{2.27}$$

With respect to the general finite mixture model formulation, the *a priori* parameters, $\boldsymbol{\xi}$ contains the elements of the component means $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$, shown by Equation 2.28.

$$\boldsymbol{\xi} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K) \tag{2.28}$$

Thus, the entire unknown parameters, $\boldsymbol{\Psi}$, belonging to a GMM is expressed by Equation (2.29).

$$\boldsymbol{\Psi} = (\pi_1, \ldots \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K) \tag{2.29}$$

In the formation of a model, data is provided and the goal is to assume a distribution and estimate parameters that define that distribution. For the GMM, these parameters that require estimation are seen in Equation 2.29. The most common method to estimating these parameters is to use the Maximum Likelihood Estimation [54].

2.5 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a parametric method often used for classification. Since MLE is a parametric method it is assumed that samples are drawn from some distribution that may be described by a known model. This model has parameters that may be estimated using statistical inference, i.e. mean and variance for a Gaussian. After estimating the parameters from samples, an estimated distribution may be described using the estimated parameters. From the estimated distribution, likelihood values may be determined and utilized in decisions during classification. The formulation of MLE and its application to classification may be found in finer detail in [56, 57] which is the basis to this section's mathematical representation and derivations.

2.5.1 MLE Univariate Formulation

For this section, the *univariate* case of MLE is derived for conceptual understanding prior to formalizing multivariate and generalized forms. Let $\mathbf{y} = \{y_1, \ldots, y_N\}$ be N independent and identically distributed (*iid*) samples. It is assumed that the samples are drawn from a known distribution belonging to a probability density $f(\mathbf{y} \mid \boldsymbol{\Psi})$ with parameters $\boldsymbol{\Psi}$:

$$y_j \sim f(\mathbf{y} \mid \boldsymbol{\Psi}) \tag{2.30}$$

The goal of MLE is to find estimators of the assumed model parameters $\boldsymbol{\Psi}$, such that the sampling of y_j from $f(\mathbf{y} \mid \boldsymbol{\Psi})$ becomes as likely as possible as shown in Equation 2.31.

$$\hat{\boldsymbol{\Psi}} = \underset{\boldsymbol{\Psi}}{\operatorname{argmax}}\, f(\mathbf{y} \mid \boldsymbol{\Psi}) \tag{2.31}$$

Since y_j are *iid* samples from $\mathbf{y}$, the likelihood of $\mathbf{y}$ given $\boldsymbol{\Psi}$ may be expressed as a product of likelihoods of the individual points as described in Equation 2.32.

$$l(\boldsymbol{\Psi}) \equiv f(\mathbf{y} \mid \boldsymbol{\Psi}) = \prod_{j=1}^{N} f(y_j \mid \boldsymbol{\Psi}) \tag{2.32}$$

However, the goal is to find $\hat{\boldsymbol{\Psi}}$ that makes $\mathbf{y}$ most likely to be drawn and this is denoted as $l(\boldsymbol{\Psi} \mid \mathbf{y})$. Furthermore, the log of the likelihood is used as a computational simplification, and is especially appealing to exponential families. The *log likelihood* is defined as:

$$\mathcal{L}(\boldsymbol{\Psi} \mid \mathbf{y}) = \log l(\boldsymbol{\Psi} \mid \mathbf{y}) = \log \prod_{j=1}^{N} f(y_j \mid \boldsymbol{\Psi}) \tag{2.33}$$

Given that the log is a monotonic function, maximizing the log likelihood will also result in maximizing the density function as seen in Equation 2.34.

$$\hat{\boldsymbol{\Psi}} = \underset{\boldsymbol{\Psi}}{\operatorname{argmax}}\{\log f\left(\mathbf{y} \mid \boldsymbol{\Psi}\right)\} = \underset{\boldsymbol{\Psi}}{\operatorname{argmax}}\{f\left(\mathbf{y} \mid \boldsymbol{\Psi}\right)\} \tag{2.34}$$

The remaining process of MLE is to derive estimators for the parameters based on the assumed model used to describe the distribution of the samples. There are many models that may be selected but in the interest of brevity only Gaussian estimators are discussed.

2.5.2 MLE Gaussian Estimators - Univariate

Using an univariate Gaussian distribution as a model is to assume that the samples are expected to be normally distributed with mean μ and variance σ^2, also commonly denoted as $\mathcal{N}\left(\mu, \sigma^2\right)$. The univariate Gaussian density function for a sample, y, can be evaluated by Equation (2.35).

$$f\left(y; \mu, \sigma\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left(y-\mu\right)^2}{2\sigma^2}\right], -\infty < x < \infty \tag{2.35}$$

Given a sample $\mathbf{y} = \{y_1, \ldots, y_N\}$ with size N, where $y_j \sim \mathcal{N}\left(\mu, \sigma^2\right)$ and $j = \{1, \ldots, N\}$, the log likelihood of $\mathbf{y}$ may be calculated using Equation (2.36).

$$\mathcal{L}\left(\mu, \sigma \mid \mathbf{y}\right) = \log \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_j-\mu\right)^2}{2\sigma^2}\right) \tag{2.36}$$

With further simplification, as detailed in Appendix A.1, the log likelihood of a Gaussian sample can be determined by Equation (2.37).

$$\mathcal{L}\left(\mu,\sigma\,|\,\mathbf{y}\right) = -\frac{N}{2}\log\left[2\pi\right] - N\log\left[\sigma\right] - \sum_{j=1}^{N}\frac{\left(y_j-\mu\right)^2}{2\sigma^2} \tag{2.37}$$

To find estimators $\hat{\mu}$ and $\hat{\sigma}^2$ that maximize the log likelihood $\mathcal{L}\left(\mu,\sigma\,|\,\mathbf{y}\right)$, we find roots of the partial derivatives with respect to μ and σ.

$$\frac{\partial}{\partial\mu}\mathcal{L}\left(\mu,\sigma|\mathbf{y}\right) = \frac{\partial}{\partial\mu}\left[-\frac{N}{2}\log\left(2\pi\right) - N\log\left(\sigma\right) - \frac{\sum_{j=1}^{N}\left(y_j-\mu\right)^2}{2\sigma^2}\right] = 0 \tag{2.38}$$

$$\frac{\partial}{\partial\sigma}\mathcal{L}\left(\mu,\sigma|\mathbf{y}\right) = \frac{\partial}{\partial\sigma}\left[-\frac{N}{2}\log\left(2\pi\right) - N\log\left(\sigma\right) - \frac{\sum_{j=1}^{N}\left(y_j-\mu\right)^2}{2\sigma^2}\right] = 0 \tag{2.39}$$

Solving for μ and σ respectively will yield estimators:

$$\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} y_j \tag{2.40}$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{j=1}^{N}\left(y_j-\hat{\mu}\right)^2 \tag{2.41}$$

Using the process outlined in this subsection, the same can be applied to estimating the parameters of a multivariate Gaussian distribution.

2.5.3 MLE Gaussian Estimators - Multivariate

In many applications there are multiple inputs and outputs, thus requiring data to be modeled in a multivariate form. In multivariate form, sample data $\mathbf{Y}$ is represented as a data matrix with M variables and N observations as shown by Equation 2.42.

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}_{1,1} & \mathbf{Y}_{1,2} & \cdots \mathbf{Y}_{1,M} \\ \mathbf{Y}_{2,1} & \mathbf{Y}_{2,2} & \cdots \mathbf{Y}_{2,M} \\ \vdots & & \\ \mathbf{Y}_{N,1} & \mathbf{Y}_{N,2} & \cdots \mathbf{Y}_{N,M} \end{bmatrix} \tag{2.42}$$

The multivariate Gaussian distribution or multivariate normal distribution (MVN) may be uniquely defined by its mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The mean vector $\boldsymbol{\mu}$ is composed of the expected values of the individual columns of $\mathbf{Y}$, as explained by Equation (2.43).

$$E\left[\mathbf{Y}\right] = \boldsymbol{\mu} = \left[\mu_1, \cdots, \mu_M\right]^T \tag{2.43}$$

The covariance of column-vector $\mathbf{y}_j$ and row-vector $\mathbf{y}_k$ is defined by Equation (2.44), where σ_j^2 is the variance of $\mathbf{y}_j$.

$$\text{Cov}\left(\mathbf{y}_j, \mathbf{y}_k\right) \equiv \sigma_{jk} = E\left[\left(\mathbf{y}_j - \mu_j\right)\left(\mathbf{y}_j - \mu_k\right)\right] = E\left[\mathbf{y}_j \mathbf{y}_k\right] - \mu_j \mu_k \tag{2.44}$$

All of the variances and covariances are represented as the $M \times M$ covariance matrix $\mathbf{\Sigma}$ and can be visualized as Equation (2.45).

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,M} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,M} \\ \vdots & & & \\ \sigma_{N,1} & \sigma_{N,2} & \cdots & \sigma_M^2 \end{bmatrix} \tag{2.45}$$

The covariance matrix is assumed to be a symmetric positive semi-definite where the diagonals terms are the variances and the off-diagonal terms are the covariances. In vector-matrix form the covariance matrix can be evaluated by using Equation (2.46).

$$\mathbf{\Sigma} \equiv \mathrm{Cov}\left(\mathbf{Y}\right) = E\left[\left(\mathbf{Y} - \boldsymbol{\mu}\right)\left(\mathbf{Y} - \boldsymbol{\mu}\right)^T\right] = E\left[\mathbf{Y}\mathbf{Y}^T\right] - \boldsymbol{\mu}\boldsymbol{\mu}^T \tag{2.46}$$

For a MVN it is common that variables will have unequal variances and may exhibit some correlations. Therefore, the Mahalanobis distance is used to calculate the distance between an observation $\mathbf{y}_j$ and the mean $\boldsymbol{\mu}$ in standardized units as defined by Equation (2.47).

$$d\left(\mathbf{y}_j, \boldsymbol{\mu}\right) = \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \mathbf{\Sigma}^{-1}\left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{2.47}$$

This distance measure is used to define the probability density function of a MVN or as seen in Equation (2.48), where $\mathbf{y}_j \sim \mathcal{N}_M\left(\boldsymbol{\mu}, \mathbf{\Sigma}\right)$, $|\cdot|$ is the determinant operator and M

is the dimension cardinality of $\mathbf{y}_j$.

$$f\left(\mathbf{y}_j; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\left(2\pi\right)^{M/2} \left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right] \tag{2.48}$$

The log likelihood of a multivariate Gaussian distribution is shown by Equation (2.50).

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \log \prod_{j=1}^{N} f\left(\mathbf{y}_j \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \tag{2.49}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \log \prod_{j=1}^{N} \frac{1}{\left(2\pi\right)^{M/2} \left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right] \tag{2.50}$$

Using Equation (2.50), let $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ be the precision matrix (inverse of the covariance matrix) and further simplification (Appendix A.2) yields the log likelihood of the MVN to be defined by Equation (2.51).

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = -\frac{N \cdot M}{2} \log\left(2\pi\right) + \frac{N}{2} \log\left|\boldsymbol{\Lambda}\right| - \frac{1}{2} \sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{2.51}$$

To find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}$ using MLE is to solve:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \log f\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = 0 \text{ and } \frac{\partial}{\partial \boldsymbol{\Lambda}} \log f\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = 0 \tag{2.52}$$

Solving for the roots yield the estimators $\hat{\boldsymbol{\mu}} \in \mathbb{R}^M$ and $\hat{\boldsymbol{\Sigma}}$ to be found by using Equations (2.53) and (2.54), with further derivation of these Equations be found in Appendix A.3.

Note that $\hat{\boldsymbol{\mu}}$ has the dimensionality of $[1 \times M]$ and $\hat{\boldsymbol{\Sigma}}$ is $[M \times M]$.

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{y}_j \tag{2.53}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{j=1}^{N} (\mathbf{y}_j - \hat{\boldsymbol{\mu}}) (\mathbf{y}_j - \hat{\boldsymbol{\mu}})^T \tag{2.54}$$

It has been shown that MLE may be used to estimate parameters for probability density-based distributions; specifically univariate and multivariate Gaussians. When a mixture of distributions (mixture model) is used often a closed form solution for the model estimators is not feasible. A method to solving for estimators for a mixture model is to use an iterative algorithm such as the EM algorithm.

2.6 Expectation-Maximization Algorithm

Expectation Maximization (EM) is a well known algorithm [58] that is an iterative procedure used to compute the Maximum Likelihood Estimation (MLE) with missing or hidden data. The basic EM iteration involves two processes: The E(xpectation)-step and the M(aximization)-step. In the E-step, the missing data is estimated with respect to the observed data and current estimate of the model parameters. In the M-step, the likelihood of the data is maximized assuming that the missing data is known. These steps are iterated until convergence. The convergence of EM is certain because likelihood is guaranteed to increase with each iteration. EM may be derived in a few manners. Two of those are the direct approach and the incomplete data approach.

2.6.1 EM - Direct Approach

A direct approach of EM, as defined in [37], is to compute the MLE by solving the likelihood equation. The log likelihood of $\mathbf{\Psi}$ for a K component mixture is formed from a observed random sample $\mathbf{y} = \left(y_1^T, \ldots, y_N^T\right)^T$ is given by Equation (2.55).

$$\begin{aligned} \mathcal{L}(\mathbf{\Psi}) = \log l(\mathbf{\Psi}) &= \sum_{j=1}^{N} \log f(\mathbf{y}_j; \mathbf{\Psi}) \\ &= \sum_{j=1}^{N} \log \left[\sum_{i=1}^{K} \pi_i f_i(\mathbf{y}_j; \boldsymbol{\theta}_i) \right] \end{aligned} \tag{2.55}$$

Therefore, the computation of the MLE of $\mathbf{\Psi}$ requires solving the log likelihood equation

$$\frac{\partial \mathcal{L}(\mathbf{\Psi})}{\partial \mathbf{\Psi}} = 0 \tag{2.56}$$

After manipulation [59], the MLE of $\mathbf{\Psi}$, $\hat{\mathbf{\Psi}}$, is:

$$\hat{\pi}_i = \sum_{j=1}^{N} \frac{\tau_i\left(\mathbf{y}_j; \hat{\mathbf{\Psi}}\right)}{N} \quad (i = 1, \ldots, K) \tag{2.57}$$

and

$$\sum_{i=1}^{K} \sum_{j=1}^{N} \tau_i\left(\mathbf{y}_j; \hat{\mathbf{\Psi}}\right) \frac{\partial}{\partial \boldsymbol{\xi}} \log f_i\left(\mathbf{y}_j; \hat{\boldsymbol{\theta}}_i\right) = 0 \tag{2.58}$$

where

$$\tau_i\left(\mathbf{y}_j; \hat{\mathbf{\Psi}}\right) = \frac{\pi_i f_i(\mathbf{y}_j; \boldsymbol{\theta}_i)}{\sum_{h=1}^{K} \pi_h f_h(\mathbf{y}_j; \boldsymbol{\theta}_h)} \tag{2.59}$$

is the posterior probability that $\mathbf{y}_j$ belongs to the ith component mixture and $\boldsymbol{\xi}$ contains all the *a priori* parameters in $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$. The direct approach listed in this section may used if the data labels of the observations are known, or in other words, each to which mixture each observation belongs is known. However, it is usually the case that the data labels are not known and must also be estimates. Therefore, the EM algorithm may formulated as an incomplete-data problem.

2.6.2 EM - Formulation as Incomplete-Data Problem

A typical approach to using the EM is to formulate the problem such that the observed data vector is assumed to be incomplete. The incomplete data is assumed to be the component-label vectors $\mathbf{z}_1, \ldots, \mathbf{z}_n$ that indicate which component $\mathbf{y}_j$ belongs. Thus, $\mathbf{z}_j$ is a K-dimensional vector with $z_{ij} = 1$ or 0 and $(i = 1, \ldots, K; j = 1, \ldots, N)$. Then the complete data may be expressed as

$$\mathbf{y}_c = \left(\mathbf{y}^T, \mathbf{z}^T\right)^T \tag{2.60}$$

where

$$\mathbf{z} = \left(\mathbf{z}_1^T, \ldots, \mathbf{z}_n^T\right)^T \tag{2.61}$$

The log likelihood of the complete data is given by

$$\mathcal{L}_c\left(\mathbf{\Psi}\right) = \sum_{i=1}^{K} \sum_{j=1}^{N} z_{ij} \left[\log \pi_i + \log f_i\left(\mathbf{y}_j; \boldsymbol{\theta}_i\right)\right] \tag{2.62}$$

E-step

The E-step requires the calculation of the conditional expectation of $\mathcal{L}_c(\boldsymbol{\Psi})$ for the (k)th iteration:

$$Q\left(\boldsymbol{\Psi};\boldsymbol{\Psi}^{(t)}\right)=\tau_i\left(\mathbf{y}_j;\boldsymbol{\Psi}^{(t)}\right) \tag{2.63}$$

Specifically, the E-step computes the posterior probabilities that the jth member of the sample with observed value $\mathbf{y}_j$ belongs to the ith component. The posterior probabilities may be calculated as

$$\tau_i\left(\mathbf{y}_j;\boldsymbol{\Psi}^{(t)}\right)=\frac{\pi_i^{(t)}f_i\left(\mathbf{y}_j;\boldsymbol{\theta}_i^{(t)}\right)}{\sum_{h=1}^{g}\pi_h^{(t)}f_h\left(\mathbf{y}_j;\boldsymbol{\theta}_h^{(t)}\right)} \tag{2.64}$$

M-step

The M-step is the maximization of the conditional expectation of $Q\left(\boldsymbol{\Psi};\boldsymbol{\Psi}^{(t)}\right)$ to give an updated estimate for $\boldsymbol{\Psi}^{(t+1)}$. The estimates for mixing proportions $\pi_i^{(t+1)}$ and the unknown parameters to the component densities $\boldsymbol{\xi}^{(t+1)}$ are done independently of each other. The estimate of π_i for the ith iteration is equivalent to the sum of the posterior probabilities of membership of each observation $\mathbf{y}_j$ belonging to the ith mixture component, as seen as

$$\pi_i^{(t+1)}=\frac{1}{N}\sum_{j=1}^{N}\tau_i\left(\mathbf{y}_j;\boldsymbol{\Psi}^{(t)}\right) \tag{2.65}$$

Updating $\boldsymbol{\xi}$ on the $(t+1)$th iteration of the EM involves finding the root of:

$$\sum_{i=1}^{K}\sum_{j=1}^{N}\tau_i\left(\mathbf{y}_j;\boldsymbol{\Psi}^{(t)}\right)\frac{\partial}{\partial\boldsymbol{\xi}}\log f_i\left(\mathbf{y}_j;\boldsymbol{\theta}_i\right)=0 \tag{2.66}$$

For most distributions there exists a closed form of solution of Equation 2.66. The final estimate of $\mathbf{\Psi}$ is found by repeating E and M steps until very little change in likelihood values are seen.

$$\mathcal{L}\left(\mathbf{\Psi}^{(t+1)}\right) - \mathcal{L}\left(\mathbf{\Psi}^{(t)}\right) < e \tag{2.67}$$

where e is arbitrarily selected based on desired convergence criteria. Additionally it has been proven by Dempster et al. [58] that the likelihood function $\mathcal{L}(\mathbf{\Psi})$ does not decrease with each iteration, as summarized by the following inequality

$$\mathcal{L}\left(\mathbf{\Psi}^{(t+1)}\right) \geq \mathcal{L}\left(\mathbf{\Psi}^{(t)}\right) \tag{2.68}$$

In this section, EM has been explained as a general form to solving estimators of mixture models. When applying EM to a specific mixture, the E-step and M-step need to be derived to reflect the chosen distribution estimators. As an example of this EM process, determining the estimators for a mixture of Gaussians is considered.

2.6.3 EM - Gaussian Mixture Models

Applying EM to a mixture of Gaussians is to use the general form discussed previously and substituting in the density function $f_{\mathcal{N}}$ as defined in Equation 2.27. For the E-step, the posterior probabilities are calculated as:

$$\tau_i\left(\mathbf{y}_j; \mathbf{\Psi}^{(t)}\right) = \frac{\tau_i^{(t)} f_{\mathcal{N}}\left(\mathbf{y}_j; \boldsymbol{\theta}_i^{(t)}\right)}{\sum\limits_{h=1}^{K} \pi_h^{(t)} f_{\mathcal{N}}\left(\mathbf{y}_j; \boldsymbol{\theta}_h^{(t)}\right)} \tag{2.69}$$

Then for the M-step, the mixing weights are estimated the same as the general case:

$$\pi_i^{(t+1)} = \frac{1}{N} \sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right) \tag{2.70}$$

Calculating the estimates of the model parameters of the Gaussians is to substitute $f_{\mathcal{N}}$ in the general formulation and find the roots of

$$\sum_{i=1}^{K} \sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right) \frac{\partial}{\partial \boldsymbol{\xi}} \log f_{\mathcal{N}} \left(\mathbf{y}_j; \boldsymbol{\theta}_i \right) = 0 \tag{2.71}$$

After solving the estimators $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ for the $(t+1)^{th}$ iteration may be calculated using:

$$\boldsymbol{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right) \mathbf{y}_j}{\sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right)} \tag{2.72}$$

$$\boldsymbol{\Sigma}_i^{(t+1)} = \frac{\sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right) \left(\mathbf{y}_j - \boldsymbol{\mu}_i^{(t+1)} \right) \left(\mathbf{y}_j - \boldsymbol{\mu}_i^{(t+1)} \right)^T}{\sum_{j=1}^{N} \tau_i \left(\mathbf{y}_j; \boldsymbol{\Psi}^{(t)} \right)} \tag{2.73}$$

Using a training set of reduced feature vectors, $\mathbf{Y}$, the EM algorithm is used to estimate parameters for a GMM. The training set only contains feature vectors from the system during known *normal* operation. One-class classification techniques are then used to define a new feature vector's belonging to the learned GMM, because the GMM.

2.7 One-Class Classification with GMM

One-class classification is generally formalized using distance or probability measures. An object $\mathbf{z}$ is accepted to the target class if it satisfies the appropriate indicator function I. For distance-based measures, new objects are accepted when the distance to the target class is smaller than a threshold value θ_d:

$$f(\mathbf{z}) = I\left(d\left(\mathbf{z}|\omega_T\right) < \theta_d\right) \tag{2.74}$$

and for probability-based measures, the object must have a probability larger than θ_p:

$$f(\mathbf{z}) = I\left(p\left(\mathbf{z}|\omega_T\right) > \theta_p\right) \tag{2.75}$$

where ω_T is set of data from the target and information from any other outlier distributions is not used [25].

In one-class classification, information from only a single class is used or also known as the *target* class. A boundary is defined around the target class such that those input vectors within the boundary are classified as the target class and others are considered non-target [25]. The model for the *target* class is estimated using a GMM, which is a probability density function and thus its output is the likelihood of the input's belonging to the GMM.

The Mahalanobis distance is used to quantify a feature vector's distance from the learned GMM. The Mahalanobis requires the inverse of the covariance matrix, where instead the Cholesky decomposition of the covariance matrix ($\hat{\mathbf{\Sigma}}$) can be used [36]. The

Cholesky decomposition factorizes the covariance matrix into an upper triangular matrix for improved computational efficiency of the inverse calculation. Additionally, using the Cholesky form of the covariance reduces the Mahalanobis distance to the Euclidean norm. An individual input vector, $\mathbf{y}_j$, distance to each mixture of a GMM is calculated by using Equation (2.76), where $\hat{\boldsymbol{\mu}}_i$, $\hat{\mathbf{R}}_i$ are the estimated mean vector and Cholesky form of the covariance matrix for the i^{th} mixture learned using the EM algorithm.

$$d_{i,j}\left(\mathbf{y}_j; \hat{\boldsymbol{\mu}}_i, \hat{\mathbf{R}}_i\right) = \|(\mathbf{y}_j - \hat{\boldsymbol{\mu}}_i)\left(\hat{\mathbf{R}}_i\right)^{-1}\|^2 \tag{2.76}$$

The likelihood of an input vector, $\mathbf{y}_j$, belonging to the i^{th} mixture is calculated by using Equation (2.77) and belonging to each mixture concatenated into a single vector Equation (2.78).

$$l_{i,j}\left(d_{i,j}; \hat{\mathbf{R}}_i\right) = \frac{2\pi^{M/2}}{det\left(\hat{\mathbf{R}}_i\right)} \exp\left(-\frac{1}{2}d_i\right) \tag{2.77}$$

$$\boldsymbol{l}_j = \{l_{1,j}, l_{2,j}, \ldots, l_{K,j}\} \tag{2.78}$$

Using the estimated mixture proportions $\hat{\boldsymbol{\pi}} = \{\hat{\pi}_1, \ldots, \hat{\pi}_K\}^T$ and the likelihood values ($\mathcal{L}$), the log-likelihood of $\mathbf{y}_j$ belonging to the GMM is determined using Equation (2.79).

$$\mathcal{L}_j = \log\left[\boldsymbol{l}_j \hat{\boldsymbol{\pi}}\right] \tag{2.79}$$

Using the log-likelihood value, $\mathcal{L}_j$, a binary decision for one-class classification is achieved via thresholding. A threshold value is determined statistically from a training set. The threshold value is defined using Equation (2.80), where $\mu_{\mathcal{L}}$ and $\sigma_{\mathcal{L}}$ are the mean and variance of the log-likelihood of the training set. The parameter λ is introduced to affect the rate of acceptance/rejection of outliers. A feature vector $\mathbf{y}_j$ is considered belonging to the model ('1') by evaluating the indicator function in Equation (2.81).

$$\mathcal{L}_{\tau} = \mu_{\mathcal{L}} - \lambda\sqrt{\sigma_{\mathcal{L}}} \tag{2.80}$$

$$I_{\mathcal{L}}\left(\mathbf{y}\right) := \begin{cases} 1 & \text{if } \log \mathcal{L} < \mathcal{L}_{t\tau} \\ 0 & \text{otherwise} \end{cases} \tag{2.81}$$

Using the methods provided in this section, we next present some performance metrics for binary decisions.

2.7.1 Binary Decision Performance Metrics

Binary decisions are commonly required for one/two class classification problems. The accuracy of a binary decision may be quantified by various statistical measures as discussed by Metz [60] and also by Murphy [55]. Most of these measures are ratios formed based on combinations of the number of *true positive* (TP), *true negative* (TN), *false positive* (FP), and *false negative* (FN). Table 2.1 provides the definitions for TP, TN, FP, and FN as well as some other intermediate metrics. From Table 2.1, many different statistical measures may be formed, a few have been provided in Equations (2.82)-(2.86); *true positive rate* (TPR) or *sensitivity*, *specificity* (SPC) or *neagtive rate*, *positive predictive value* (PPV) or *precision*, *negative predictive value* (NPV), *false positive rate* (FPV)

Table 2.1: Quantities that may be derived from a confusion matrix.

		Truth		
		1	0	Σ
Estimate	1	TP	FP	$\hat{N}_+ = TP + FP$
	0	FN	TN	$\hat{N}_- = FN + TN$
	Σ	$N_+ = TP + FN$	$N_- = FP + TN$	$N = TP + FP + FN + TN$

or *false alarm rate*, and *accuracy* (ACC). These statistical measures can be combined to compute a loss function. This loss function may be used to select a threshold (τ) to minimize the loss function.

$$TPR = \frac{TP}{N_+} = \frac{TP}{TP + FN} \tag{2.82}$$

$$SPC = \frac{TN}{N_-} = \frac{TP}{FP + TN} \tag{2.83}$$

$$PPV = \frac{TP}{\hat{N}_+} = \frac{TP}{TP + FP} \tag{2.84}$$

$$NPV = \frac{TN}{\hat{N}_-} = \frac{TN}{FN + TN} \tag{2.85}$$

$$FPR = \frac{TN}{\hat{N}_-} = \frac{TN}{FN + TN} \tag{2.86}$$

$$ACC = \frac{TP + TN}{N} = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.87}$$

The method proposed in this section defines generic approach(es) to preprocessing, feature vector creation, learning/using GMMs, and one-class classification for time-series data. However, actual preprocessing techniques, feature vector creation, and performance metrics should be selected to suit the application domain and its corresponding data. The next chapters provide two different applications and their specific processing choices that are supported by empirical evaluation of classification performance metrics.

Chapter 3

RF Power Generator

Presented in this chapter is an application of the proposed ONDS to radio frequency (RF) power generators used in the semiconductor industry. A RF power generator supplies an RF power signal that is output to a load. To achieve and maintain a desired RF power signal or *setpoint*, a closed-loop control system is used. The particular RF generators used in our experimental setup are those from MKS ENI Products' Regulus series and their high-level control system is depicted in Figure 3.1. From Figure 3.1, it is seen that the RF power generator includes a variable voltage supply, pre-amplifier circuitry, RF power amplifier (PA) module(s), RF sensor, various other sensors, and a controller. Basic control of the RF generator power output employs negative feedback from various sensors (such as *forward* power and *reverse* power) to achieve and sustain the external setpoint. The controller yields a *RF drive* for pre-amplification and controls the *rail setpoint* of the variable voltage supply. After pre-amplification, the power modules use the pre-amplified RF drive signal and power from the variable voltage supply (*PA voltage* and *PA current*) to generate the RF power signal [61]. The output of the RF power generator is fed through a *matching network* prior to connection to the load. The matching network is

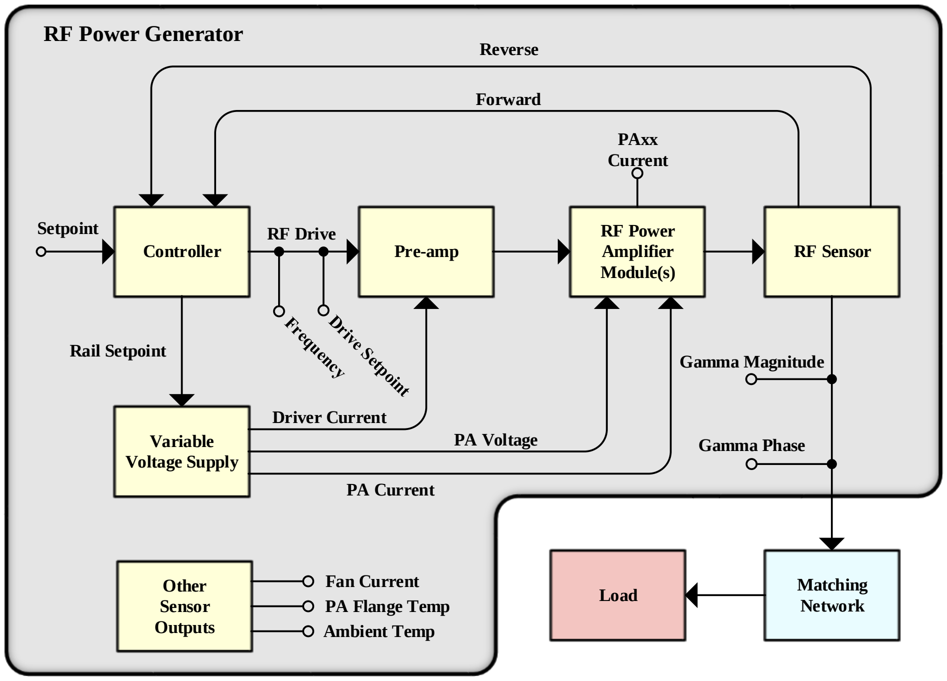


Figure 3.1: Generalized block diagram for closed-loop system for an RF power generator with a subset of input/output signals generated during operation.

used to match the impedance of the generator output to the impedance of the plasma load. Impedance matching is required to provide maximum power transfer and to prevent reflected power on the transmission lines [62]. In order to control and model the behavior of an RF generator: understanding these variables is essential.

3.1 System Variables

An RF power generator, even from a high-level abstraction (Figure 3.1), is a complex electrical system. There exists the potential for numerous inputs, outputs, internal, and derived system variables. The under-lying fundamental understanding of the dynamics of these system variables is required to properly evaluate systemic health. Expert knowledge of the system variables and their expected dynamics facilitates the realization of a procedure to generate a system signature. Consequently, the RF generator domain experts generated a subset of system variables to be used to create a system signature. Within this subset of variables are three categories consisting of vital variables, environmental variables, and variables that indicate system life. With the exception of the lifetime indicators, the other system variables may be seen in Figure 3.1.

The vital system variables (Table 3.1) are *directly* related to the RF power conversion process and plasma load impedance. For example, a subtle decrease in the RF power amplifier efficiency can be detected by increases in control system actuators such as the *RF Drive Setpoint* or *Rail Setpoint*. The vital variables are also available from filtered data at very high sampling rates (>100kHz), enabling the detection of electrical transients or instabilities. The environmental variables (Table 3.2) are *indirectly* related to generator health. For example, high PA transistor flange temperature can be indicative of an environmental problem (e.g. restricted coolant flow). Finally, the lifetime variables (Table 3.3) can be correlated with certain wear-out mechanisms (e.g. contact wearout, metal migration, solder fatigue, and electrolytic capacitor aging). The vital and environmental variables specific to the MKS Regulus series RF power generator are briefly defined in Tables 3.1 and 3.2. A few of the lifetime variable are also listed in Table 3.3, while others have been left out for proprietary reasons.

Based on expert knowledge of the RF power generator system, there are some expected pattern dynamics among some of the variables. A system signature that does not exhibit these expected pattern dynamics is a target for consideration of a non-normal behavior. For example, it is expected that the *Rail Setpoint* will always match the *PA*

Table 3.1: MKS Regulus series RF power generator vital system variables.

Variable Name	**Description**
Setpoint	RF Power Setpoint
Forward	Forward power signal from RF output sensor
Reverse	Reflected power signal from RF output sensor
Dissipated	Power Amplifier waste heat dissipation
RF Drive Setpoint	RF amplitude actuator
RF Drive Frequency	Frequency actuator
Rail Setpoint	PA supply voltage setpoint
Gamma Magnitude	Magnitude of complex load reflection coefficient
Gamma Phase	Phase of complex load reflection coefficient
PAxx Current	Supply current to Power Amplifier xx
PA Voltage	Measured output voltage of PA power supply
Driver Current	Supply current to PA driver module

Table 3.2: MKS Regulus series RF power generator environmental system variables.

Variable Name	**Description**
Fan Current	Measured current supplied to cooling fans
Ambient Air Temp	Measured Internal Ambient air temperature
PA Flange Temp	Measured PA Power Transistor case temperature

Table 3.3: Subset of MKS Regulus series RF power generator lifetime (since build date) diagnostic indicators.

Variable Name	**Description**
AC On Time	Total hours with AC power applied
AC Cycles	Total AC turn-on events
Fault Clears	Count of faults that were cleared

Voltage variable in a healthy system. However, a discrepancy can indicate a faulty voltage sensor or power supply malfunction. Similarly, the *Setpoint* should always match the *Forward Power*. This preexisting knowledge of variable dynamics could be used to reduce the overall number of required variables in the fingerprint. For this work, the lifetime variables and environmental variables are not used as it is likely that they do not have any direction correlation to the immediate system health of the RF power generator. However all of the vital variables are left in tact to prevent erroneous elimination of significant variables. Instead, the variable reduction is done using statistical analysis techniques. Based on the provided knowledge of the targeted RF power generator architecture, the proposed health evaluation system is discussed.

3.2 ONDS on RF Power Generator

The Online Novelty Detection System (ONDS), as proposed in Chapter 2, is designed to be embedded and executed within the RF generator to provide online systemic health evaluation. The proposed ONDS for an RF power generator uses the proposed one-class classification techniques to identify if its current systemic operation belongs to what has been learned to be *normal*. The process starts by the generation of the RF power generator's system signature. This system signature for this work is referred as a *fingerprint* and it contains a time-series data collection of sensor values. The fingerprint is the input to the ONDS where its data is pre-processed and a model is learned. Using a trained *normal* model, the classifier identifies a fingerprint as *normal* if its *likelihood* of belonging to the *normal* model is within statistical margins. The model, classifier parameters, and

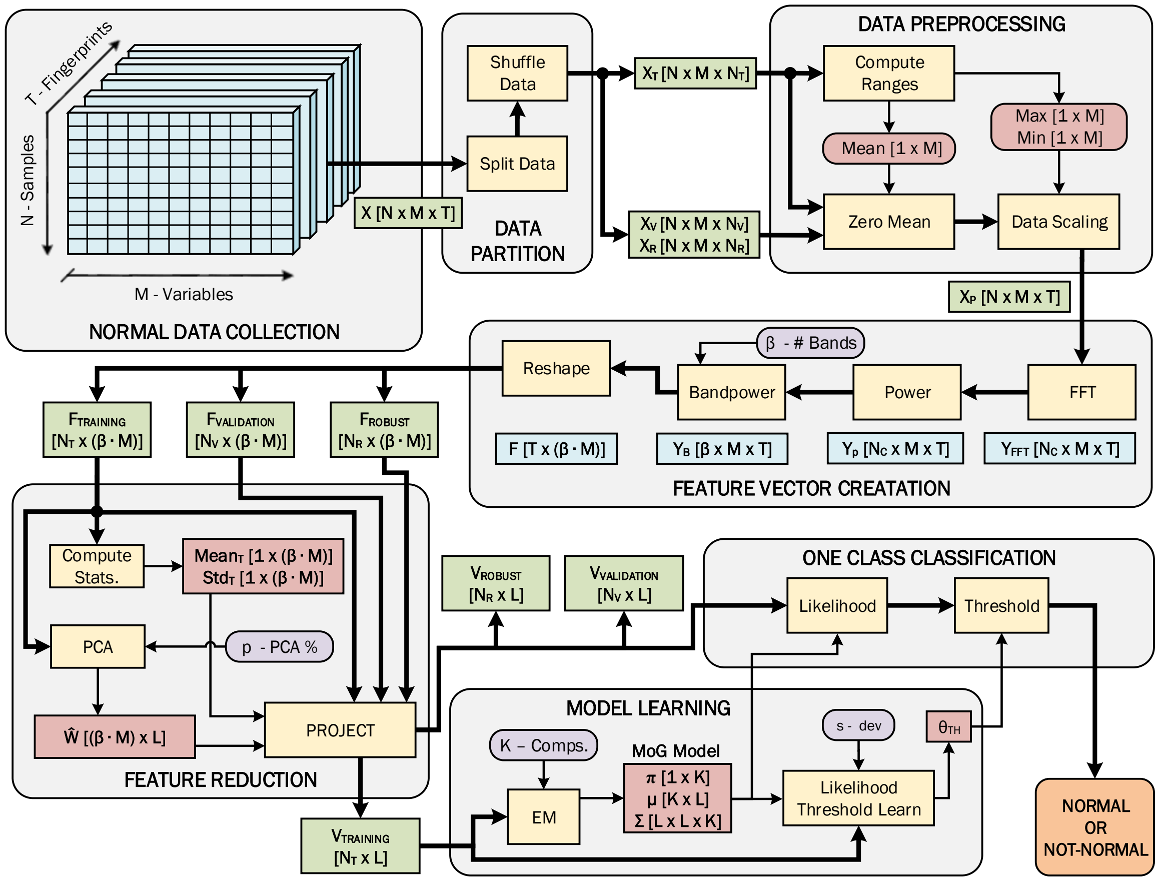


Figure 3.2: Offline Model Learning.

other algorithmic parameters are learned and/or predefined during an offline procedure as shown by Figure 3.2. The online classification stage uses similar procedures as the offline stage, as seen in the flow diagram in Figure 3.3, where a fingerprint is converted into a feature vector to be used for classification. The next section explains the data collection process as it pertains to the RF power generator application.

3.2.1 Data Collection

In order to be able to assess systemic health, a system signature needs to be collected in such a way that it holistically represents the current status of the system. Thus,

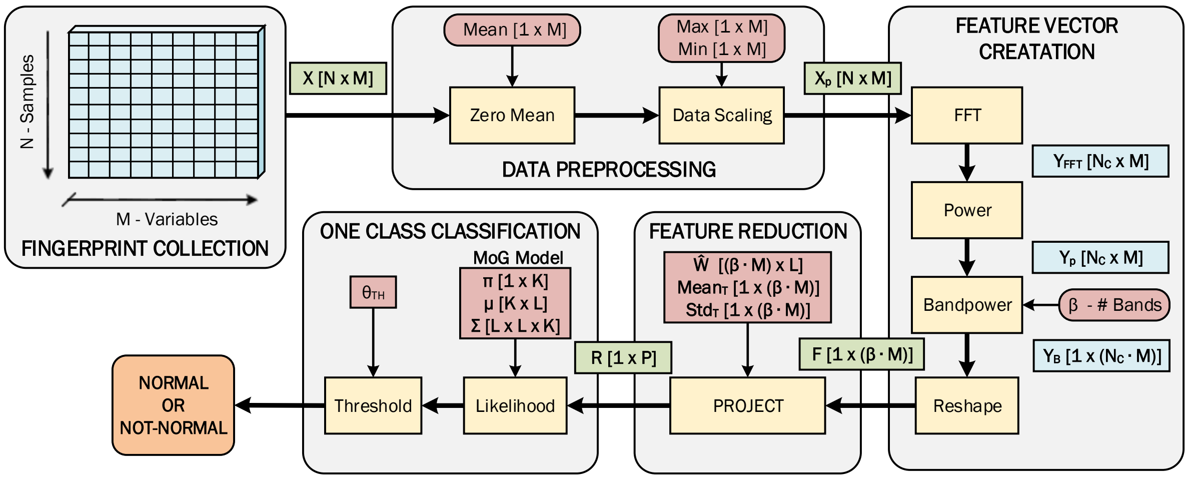
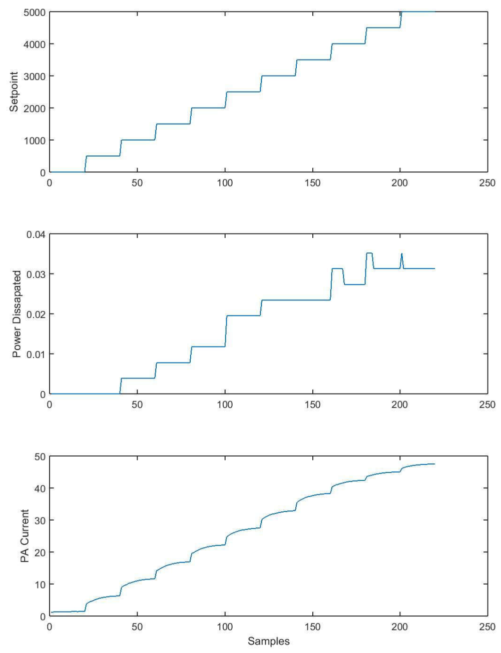


Figure 3.3: Online Classification.

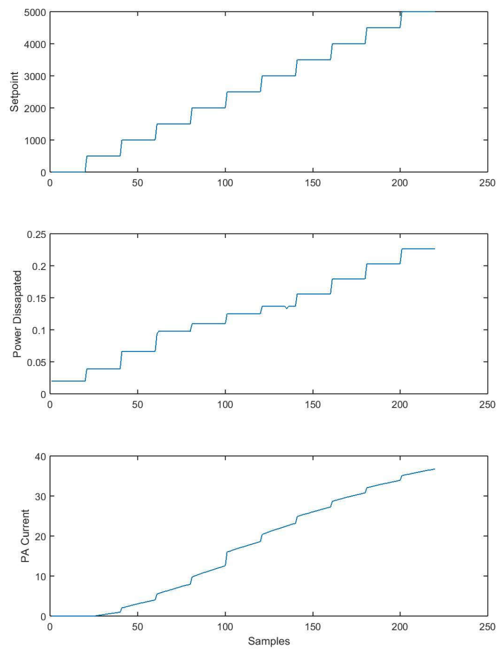
presented is a system signature or *fingerprint*, that is easily obtainable and capable of representing an RF power generator's system status. The fingerprint is a collection of multivariate time-series samples of sensor values from the RF power generator under a typical operational mode(s).

The fingerprint itself is generated by attaching the RF power generator to a desired load and sweeping the setpoint. At each setpoint, a predetermined number of samples are collected before moving to the next setpoint. The dynamic response of each system variable to the setpoint sweep becomes part of the fingerprint (e.g. the characteristic thermal response time of the PA Flange temperature when RF power is stepped). Figure 3.4 is an example of portions of two example fingerprints. In Figure 3.4, a subset of the variables have been selected (only to reduce visual complexity) and shows the subtle differences between a normal and non-normal or fault fingerprint.

The sampled data from each setpoint is concatenated into single data structure as shown in Figure 3.5. The dimensionality of the fingerprint is determined by the number of setpoints, the number of samples per setpoint and the number of system variables



(a) Normal fingerprint.



(b) Fingerprint from seeded fault.

Figure 3.4: Example raw fingerprints, showing three different variables including setpoint, power dissipation and PA current.

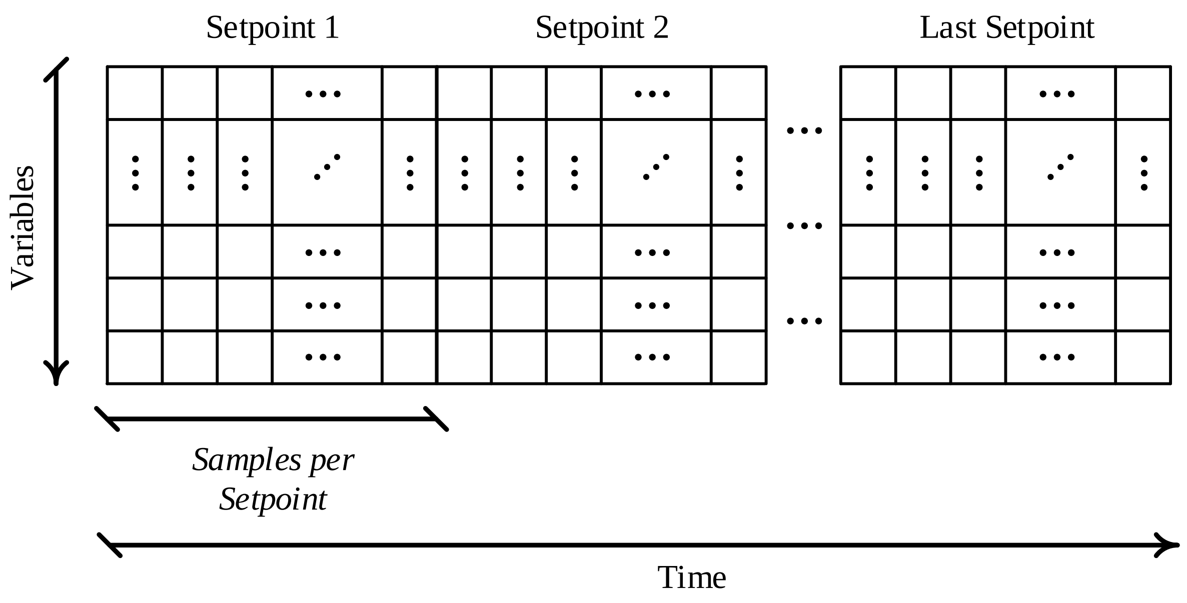


Figure 3.5: Example fingerprint.

measured. This fingperprint data collection method has been applied previously and successfully applied to fault detection in RF power generators [63, 64].

For the offline model learning stage, a precollected set of fingerprints are obtained from units that are considered to be operating normal. This restriction of using only normal fingerprints during the offline learning stage is inline with one-class classification techniques, where a model is learned for only one class. With respect to the online classification stage, the fingerprints are unclassified since they are collected online and a definitive normal/non-normal systemic health may not be available. Regardless of offline or online use, a fingerprint in its raw form poses difficulties for classification methods. Thus, some preprocessing techniques and feature extraction methods are considered.

3.2.2 Preprocessing

The specific techniques chosen have been selected according to prior knowledge that frequency features are to be extracted from the fingperprint data. The specific preprocessing techniques considered are zero-mean, normalization, and detrending. Data normalization is considered, because the fingerprint variables do not have the same data range. The inherit nature of the fingerprint, as discussed in Section 3.2.1, causes most of the variables to trend as a staircase pattern. This staircase pattern is the consequence of providing a set-point, collecting samples from the sensors, then stepping up the set-point, and repeating for the full range of setpoints. Therefore, there exists a general linear trend in the data that can have negative effects on any frequency analysis. By performing a detrending operation, this general trend is removed and may lead to more meaningful frequency analysis.

With respect to the online classification stage, all mean, maximum, and minimum values are determined during the offline stage and are provided as inputs to the online data preprocessing functional block. Once the data is preprocessed, it is ready to be transformed to extract features to create a feature vector.

3.2.3 Feature Vector Creation

During the feature vector creation process, the preprocessed fingerprint is transformed into a feature vector. Based on expert knowledge of the RF power generator system, there are some expected pattern dynamics among some of the variables. A fingerprint that does not exhibit these expected pattern dynamics is a target for consideration of non-normal behavior. For example, it is expected that the *Rail Setpoint* will always match the *PA Voltage* variable in a healthy system. However, a discrepancy can indicate a faulty voltage sensor or power supply malfunction. Similarly, the *Setpoint* should always match the *Forward Power*. This preexisting knowledge of variable dynamics supports a choice that frequency components are to be used as features.

The resulting feature vector is a single row vector that encapsulates the significant frequency components of all the variables in the fingerprint. Previous work has used the multi-dimensional Fast Fourier Transform (FFT) on each fingerprint to obtain Fourier coefficients [63, 64]. The FFT is also used in this work, but instead a single-dimensional FFT per fingerprint variable is used. The fingerprint collection process is done assuming a fixed sampling frequency. With a fixed sampling frequency, the relationship of power coefficient to frequency will be relatively consistent across fingerprints and thus the actual frequency is not as important. However, this dictates that the number of samples must remain the same for each fingerprint if any meaningful analysis is to be performed.

The fingerprint provided in the experimental setup has $N = 220$ (arbitrarily selected from a legacy system) and is zero-padded to 256 coefficients, yielding $N_{FFT} = 256$ and thus a power spectrum vector of size $[129 \times 1]$. Concatenating each variable's power spectrum vector into a single row vector would result in a vector with dimensions $[129 \cdot M \times 1]$. Depending on the RF power generator model, M is either 14 or 21. Thus, the single row vector could have a size of $[1 \times 1806]$ or $[1 \times 2709]$ respectively.

To reduce the initial number of FFT coefficients, the bandpower of the power spectrum ($\mathbf{Y}^P$) is used. The number of frequency bands used to calculate the bandpower is a predetermined value and is typically kept relatively high with $\beta \sim 80$. With eighty bands ($\beta = 80$), the fingerprint feature vector size can be reduced by $\sim 40\%$. However, even after this reduction the feature vector has the potential to have $\beta \cdot M = 80 \cdot 14 = 1120$ or $\beta \cdot M = 80 \cdot 21 = 1680$ features.

3.3 Experimental Results and Analysis

The use of Gaussian Mixture Models (GMMs) for one-class classification of systemic health for RF power generators is supported through a series of experimental tests with offline data. Prior to the application of the one-class classifier and GMM, the classifier and GMM learning procedure is validated against some standard datasets and classification techniques. The various preprocessing methods and algorithmic parameters are explored using RF power generator data with a methodical process to select those method/parameters that contribute to higher classification performances.

3.3.1 Validation of GMM and One-Class Classification

The implementation of one-class classification techniques using GMMs is applied to some standard datasets, where its classification performance is compared to some state-of-the art classifiers as well as a less complex method. Four commonly used multivariate two-class datasets are selected including: Pima Indians Diabetes (*Diabetes*), Liver-Disorders (*Liver*), Ionosphere, and Wisconsin Breast Cancer (*Cancer*). The datasets may be obtained from the UCI Machine Learning Repository and more specifically Wisconsin Breast

Table 3.4: Dataset names, corresponding sample counts, and class descriptions for the standard datasets.

Dataset	#Attributes	# Target	# Non-Target	Target	Non-Target
Diabetes	8	268	500	Negative	Positive
Liver	7	200	145	Class 2	Class 1
Ionosphere	34	225	126	Good	Bad
Cancer	10	444	239	Benign	Malignant

Cancer dataset is provided by Wolberg and Mangasarian [65, 66]. The number of attributes, sample counts, and class information for the selected datasets are summarized in Table 3.4. The *target* class is chosen to be the class that most likely represents *normal* conditions such as *no disease* or *good.* In the case of the *Liver* dataset, *Class 2* is selected as the *target* due to its higher sample count.

The four state-of-the-art classifiers chosen for comparison are; the Fuzzy Functions Support Vector Classifier (FFSVC), Improved Fuzzy Functions Support Vector Classifier (IFFSVC), a Radial Basis Function Networks (RBFN) whose centers have been chosen using Fuzzy C-Means (FCM) (FCM-RBFN) and Particle Swarm Optimization (PSO) (PSO-RBFN). The FFSVD and IFFSVD as proposed by Celikyilmaz et al. are chosen based on their demonstrated improved performances over the classical Support Vector Classifier [67, 68]. The FCM-RBFM and PSO-RBFN as proposed by Cinar and Sahin are chosen due to similarities in implementation of radial basis functions and GMMs as well as FCM-RBFM and PSO-RBFN demonstrated comparable performance with respect to the FFSVD and IFFSVD classifiers [69]. Additionally the k-Nearest Neighbor (k-NN) is selected to provide comparison to a simple machine learning algorithm [55].

For each dataset, the data is split into three subsets for training, validation and testing with respective ratios of 50%, 25% and 25%. Training data is used to train each of the classifiers, the validation data is used to cross-validate to avoid overfitting, and the test data is used to calculate performance metrics. The classification results for the four classifiers, as provided by Cinar and Sahin, are reported as classification accuracy (ACC) and standard deviation from 10 runs using the test data [69]. For each run, the data is shuffled such that different samples are considered during training, validation, and testing. The FFSVC, IFFSVC, FCM-RBFN, PSO-RBFN, and k-NN classifiers are multi-class classifiers and thus their training data contains samples from all classes.

For comparison purposes, the datasets are preprocessed with scaling to [-1,1] and organized using LIBSVM libraries written using the MATLAB® computing language [70]. For the GMM classifier, the data is further prepossessed by scaling the data such that each attribute has unit variance. The data is reduced by using PCA. For each run, the same PCA matrix is used to transform the training, validation and testing data into the PC space. The unit variance scaling and PCA matrix are calculated using only the training data. The GMM is designed to model only one class, thus samples from the *target* class, as listed in Table 3.4, are used during training. However, the validation and testing data contain samples from both the *target* and *non-target* classes.

The EM algorithm as implemented by Verbeek et al. is used to learn the GMM parameters [36]. A few algorithmic parameter selections are required including the number of mixtures (K) and the likelihood threshold. The likelihood threshold is set to be λ standard deviations lower than the mean of the training set's likelihood values. To determine the best number of mixtures and likelihood threshold a grid search was performed such that $K \in \{1, 2, \ldots, 20\}$ and $\lambda \in \{0.25, 0.5, 0.75, \ldots, 3.0\}$. The grid search was performed for each of the datasets with 10 trials and for each trial the EM algorithm was ran 25 times to help prevent a local model solution. For each EM, the accuracy (ACC), true positive rate (TPR) and specificity (SPC) are calculated for both the validation and test data (as discussed in Section 2.7.1). The accuracy of the classifier is reported from the test data considering three different EM model selection criteria. The first EM selection criteria chooses the EM model from the subset of 25 individual runs that maximizes the classification accuracy of the validation samples. Respectfully, the second and third criteria maximize the true positive rate and specificity of the validation samples. The result of each of the three EM model selection criteria are reported in Tables 3.5, 3.6 and 3.7.

For one-class classification, typically information is only available for the *target* class, thus the EM learned GMM selection should be performed using the true positive rate as it does not use any information from the *non-target* class. However, the other selection criteria (ACC and SPC) are considered as to explore which number of mixtures and threshold may allow better separability of the *non-target* from the *target* class. From Table 3.6, where TPR is used to select the GMM, it can be seen that generally lower number of mixtures produce higher classification accuracy. Whereas, for Tables 3.5 and 3.7 generally higher number of mixtures produce higher classification accuracy. This trend follows conceptual intuition that a lower number mixtures yields a more general model and accepts more samples as *target*. Consequently, a higher number of mixtures indicates a more specific model and it rejects more samples from the *target* class.

The final parameters used for EM for the standard datasets are selected from the grid search results that use the validation accuracy for GMM model selection as listed in Table 3.5. Using these parameters, the experimental procedure was repeated, however

Table 3.5: Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based on validation accuracy.

Dataset	ACC	TPR	SPC	λ	K
Diabetes	76.64 (0.30)	73.37 (0.61)	78.76 (0.90)	0.25	4
Liver	72.64 (0.22)	53.96 (0.73)	93.11 (2.52)	0.5	15
Ionosphere	91.96 (0.15)	97.22 (0.53)	86.42 (0.34)	3.0	4
Cancer	96.95 (0.52)	95.36 (1.42)	98.36 (0.90)	3.0	7

Table 3.6: Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based solely on validation true positive rate.

Dataset	ACC	TPR	SPC	λ	K
Diabetes	76.08 (0.90)	71.29 (1.29)	80.01 (0.40)	0.25	4
Liver	72.22 (0.65)	53.50 (0.90)	92.71 (1.35)	0.25	12
Ionosphere	91.89 (0.00)	97.39 (0.00)	86.15 (0.00)	0.25	1
Cancer	96.23 (0.00)	94.85 (0.00)	97.36 (0.00)	1.5	1

Table 3.7: Best classification accuracy, true positive rate, and specificity from grid search using GMM model selected based solely on validation specificity.

Dataset	ACC	TPR	SPC	λ	K
Diabetes	75.91 (0.60)	80.53 (1.42)	70.61 (1.52)	0.5	4
Liver	71.83 (0.54)	54.02 (0.63)	89.72 (2.13)	0.5	18
Ionosphere	91.89 (0.00)	97.39 (0.00)	86.15 (0.00)	0.25	1
Cancer	96.80 (0.25)	94.45 (0.89)	99.02 (0.75)	2.0	9

Table 3.8: Number of mixtures and model threshold determined from accuracy grid search, but GMM is selected based solely on validation true positive rate.

Dataset	ACC	TPR	SPC	λ	K
Diabetes	76.08 (0.60)	71.29 (1.29)	80.01 (0.40)	0.25	4
Liver	72.00 (0.68)	52.28 (0.84)	95.68 (1.57)	0.5	15
Ionosphere	88.51 (2.59)	87.73 (6.85)	88.47 (2.77)	3.0	4
Cancer	95.57 (1.26)	91.38 (2.48)	99.82 (0.31)	3.0	7

the GMM model selection is based on maximization of the validation true positive rate in accordance to one-class classification practice. The final parameters and classification accuracy for the standard datasets are listed in Table 3.8. The final GMM results from the grid search results are then compared to the other state-of-the-art classifiers results reported by Cinar and Sahin [69] and *k*-NN as listed in Table 3.9. It is noted that during the result collection process, the number of EM iterations were fixed to 40 in order to preserve a comparable number of fitness evaluations as reported by Cinar and Sahin. From Table 3.9, the GMM one-class classifier is comparable in performance to the state-of-the-art classifiers with classification accuracy within 5% of all datasets except for Ionosphere. Additionally, the GMM is at least as good as *k*-NN on all datasets. It is important to re-iterate that this comparison is comparing a one-class classifier to multi-class classifiers. Moreover, the one-class classifier model is trained using only the *target* class samples whereas the multi-class classifiers use samples from both *target* and *non-target* classes. Once the use of GMM and one-class classification has shown comparable performance results, they may be applied to the target application of systemic health evaluation of RF power generators.

Table 3.9: GMM comparison with state-of-the-art classification methods.

Dataset	PSO-RBFN	FCM-RBFN	FFSVC
Diabetes	79.27 (1.38)	79.22 (0.96)	79.30 (0.24)
Liver	70.70 (2.73)	70.12 (3.63)	76.74 (0.00)
Ionosphere	97.36 (2.03)	94.60 (0.90)	97.70 (0.00)
Cancer	99.31 (0.62)	99.81 (0.94)	98.85 (0.00)

Dataset	IFFSVC	k-NN	GMM
Diabetes	80.83 (1.17)	76.68 (1.36)	76.08 (0.60)
Liver	76.98 (0.73)	71.39 (2.55)	72.00 (0.68)
Ionosphere	99.38 (0.00)	85.88 (3.18)	88.51 (2.59)
Cancer	99.50 (0.00)	95.86 (1.11)	95.57 (1.26)

3.3.2 ONDS Application to RF Power Generator

Time-series data or *fingerprints* (discussed in Section 3.2.1), provided by MKS ENI Products, are used to train/validate a GMM as the one-class classifier. Fingerprints are obtained from MKS ENI Products' Regulus series RF power generators, where data was collected from two platforms and two models per platform. Fingerprints were collected from various individual generators during *normal* operational conditions under three different load conditions. The load conditions of *Fifty Ohms* (FO), *Short* (Sh) and *Open* (Op) are selected as an attempt to cover the full operational load conditions for the RF power generators. It is noted that different models have different variable counts based on their sensor count and architectures. Some of the variables have been removed such as environmental sensors and lifetime variables as they are generally not significantly related to the systemic health of the RF power generators. The break-down of the variable count, file count, and number of unique units per model is listed in Table 3.10. In addition to fingerprints during *normal* conditions, fingerprints from seeded fault or *non-normal* conditions from two models are provided. The *non-normal* conditions are

Table 3.10: RF power generator dataset information including variable, file, and unique unit counts.

Generator	Vars.	Used Vars.	Normal	Normal Units	Non-Normal	Non-Normal Units
LVG3527A	35	21	279	31	899	3
LVG3527B	35	21	1785	182	NA	NA
LVG3560A	35	21	328	35	428	2
LVG3560B	35	21	363	35	NA	NA
C5002	29	14	226	23	786	3
C13002	31	14	1545	158	NA	NA

conducted to closely mimic scenarios where current quality control measures have had difficulty identifying *non-normal* operation. The different seeded *non-normal* conditions for the LVG3527 and C5002 power generators are itemized below.

- LVG3527 - (Faulty Amplifier, Poor solder joints on resistors, and Poor solder joints on FETs)
- C5002 - Open diodes

Similar to the application of GMM and one-class classification of the standard datasets, the RF power generator dataset requires some algorithmic parameter selection. In the case of the RF power generator dataset, there are parameter and methodology choices to be made during preprocessing, feature vector creation, modeling/model learning, and classification. There exists a significant interdependence of these parameters and methodologies, hence an experimental procedure is used to explore the different algorithmic methodologies/parameter combinations. A grid search is performed spanning the various processing methods and their required parameters, where the different combinations tested are summarized by Table 3.11. The total number of algorithmic combinations from Table 3.11 is 2,880. Due to the large combination count, the grid search was performed

Table 3.11: Grid search method and parameter combinations explored.

Procedure	Method	Combinations
Preprocessing	Zero-Mean	None , Global, Local

Preprocessing	Scaling	None
		Local{Max, Max-Min}
		Global{Max, Max-Min}

Preprocessing	Detrend	None, Linear
Feature Creation	Band Count	10, 20, 30, 45, 60, 80
Feature Reduction	Unit Variance	No, Yes
Modeling	Mixture Count	1,2,3,4,5,7,9,11
Classification	Threshold	3.0

only for a single RF power generator model. The RF power generator model selected is the C5002 because it is a target platform for an embedded application and has seeded *non-normal* fingerprints available.

Data Partitioning

To evaluate each combination, the dataset of fingerprints from the C5002 was partitioned into training, validation, and robust subsets. The robust dataset in this case is slightly different from conventional robust data isolation. The robust data is selected such that data samples from individual RF power generator units are isolated from the training and validation. The robust dataset is constructed from data totaling 30% of the C5002 units. Furthermore, the *non-normal* fingerprints are available only on a limited number of units. Therefore, the robust fingerprints are chosen from units with *normal* data only. This robust data isolation method is used to simulate the performance of the classification algorithm on a new unit not used in the model training and validation procedures. From the remaining units, the *normal* fingerprints are split into 50% for training and 50%

for validation. The *non-normal* fingerprints are used only for validation, because one-class classification is to model only *normal* operation. Once the fingerprints are properly partitioned, the model training, validation and robust testing may continue.

Model Training

For each combination, the training *normal* fingerprints goes through a specific combination's of preprocessing, feature vector creation, and feature reduction. For feature reduction, PCA is used to transform the reduced feature features into lower dimensional space of statistically significant principal components. The result from PCA is a matrix derived from the training set that is capable of transforming feature vectors into the PC space. The reduced training set is used to train a GMM using EM, where the number of mixtures are set based on the current combination. The GMM is used to generate a log-likelihood value for each training sample. A threshold for binary classification of *normal* is calculated as λ standard deviations from the mean of the training likelihood value, where λ is fixed to be 3.0. Additionally, The EM algorithm is repeated 15 times using the same reduced features as to re-seed the starting position of the EM algorithm. The repetition is done to reduce the possibility of local solutions that can occur when using EM. The GMM parameters and likelihood values from the 15 EM runs are used to validate the GMM.

Model Validation and Robust Testing

During validation, the pre-partitioned validation fingerprints undergo the same preprocessing, feature vector creation and feature reduction as performed during training. The feature reduction is achieved by directly using the PCA matrix constructed from training feature vectors. Each of the GMM parameters and thresholds from the 15 EM runs are applied to the reduced validation features. The likelihood threshold is applied to binary classify the validation feature vectors as *normal* or *non-normal*. For each GMM parameters and respective threshold, the true positive rate (TPR) is calculated for the validation set. The GMM parameters and threshold selected are those that achieve the highest TPR, thus maximizing the model's ability to represent *normal*. Since the validation set may also contain *non-normal* fingerprints, the specificity (SPC) and accuracy (ACC) of the validation set are also recorded. Finally, the selected GMM parameters and threshold are applied to the robust fingerprint set. Only the TPR is recorded for the robust data because it only contains *normal* fingerprints. Each combination in the grid search records classification performance metrics based on the above procedures of partitioning, training, validation and testing.

Iterative Grid Search Parameter Reduction

For each combination within the search, the partitioning, training, validation, and testing is repeated 25 times. For each of the 25 trials, the partitioning is randomized to help avoid reporting performance metrics for over/under trained models. However, for each different combination in the search, the same 25 data partitioning are used as to allow for fair comparison of performance among all the combinations. As an attempt to choose the best set of processing methods and parameters, each combination's classification metrics are ranked. The sum of the *validation accuracy* (V_{ACC}) and *robust true positive rate* (R_{TPR}) are used as a score (C_{FIT}) for the ranking criteria (Equation (3.1)). It is noted that this ranking criteria is not used for modeling or validation. The C_{FIT} criteria is only used to evaluate the effectiveness of processing methods to correctly identify *normal* fingerprints as well as reject *non-normal*.

$$C_{FIT} = V_{ACC} + R_{TPR} \tag{3.1}$$

After ranking all the combinations using the C_{FIT} measure, the top 100 combinations are used to evaluate which methods and parameters are contributing to higher fitness scores. To better assess the contribution of a method/parameter combination, a weighted measure is used. Each method/parameter's percentage of occurrence is calculated and multiplied by its average fitness to create a weighted contribution ω_c, as seen in Equation (3.2).

$$\omega_c = \left(\frac{occurance}{\#samples}\right)\left(\overline{C_{FIT}}\right) \tag{3.2}$$

The ω_c measures of the first iteration of the grid search (Table 3.12) suggests that some methods/parameters are not contributing to top 100 best fitness values. Whereas other methods were closely matched in performance. Thus, the non-contributing combinations were removed and the grid search re-performed on the now smaller subset of combinations. For each iteration, a new set of randomly partitioned fingerprints are created from

Table 3.12: Grid search results for each iteration, where the percent occurrence (%), average fitness (*avg*), and weight contribution (ω_c) for each method/parameter is reported. The reported "-" refer to a method/parameter that has been removed from the search in the previous iteration.

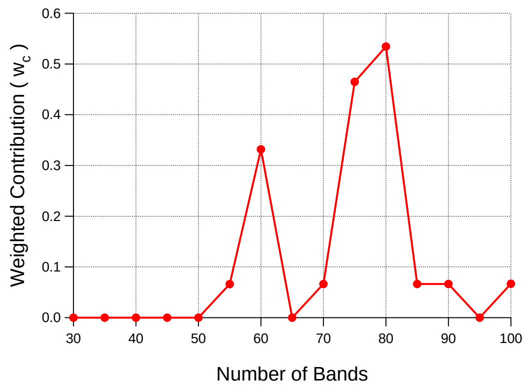
I		**Zero Mean**			**Scaling**					**Unit Var.**		**Detrend**	
		N	*L*	*G*	*N*	*ML*	*MG*	*MML*	*MMG*	*N*	*Y*	*N*	*Y*
	%	0.35	0.46	0.19	0.0	0.43	0.0	0.57	0.0	0.0	1.0	0.14	0.86
1	*avg*	1.30	1.31	1.28	0.0	1.29	0.0	1.31	0.0	0.0	1.30	1.32	1.30
	ω_c	0.46	0.60	0.24	0.0	0.56	0.0	0.75	0.0	0.0	1.30	0.19	1.12
	%	0.40	0.34	0.26	-	0.34	-	0.66	-	-	1.0	0.1	0.9
2	*avg*	1.70	1.69	1.69	-	1.69	-	1.69	-	-	1.69	1.70	1.69
	ω_c	0.68	0.58	0.44	-	0.57	-	1.12	-	-	1.69	0.17	1.52
	%	0.36	0.36	0.28	-	-	-	1.0	-	-	1.0	-	1.0
3	*avg*	1.70	1.70	1.70	-	-	-	1.70	-	-	1.70	-	1.70
	ω_c	0.61	0.61	0.48	-	-	-	1.70	-	-	1.70	-	1.70

(a) Grid search results for zero mean, scaling, unit variance, and detrending methods/parameters.

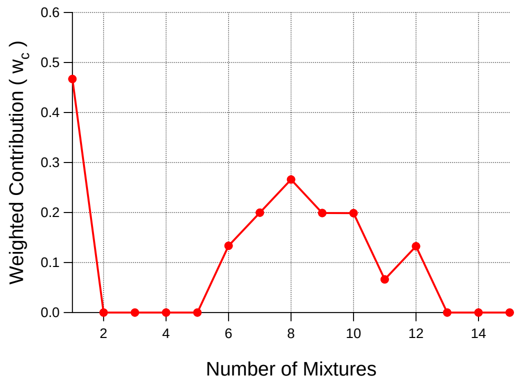
I		**Band Count**						**Mixture Count**							
		10	*20*	*30*	*45*	*60*	*80*	*1*	*2*	*3*	*4*	*5*	*7*	*9*	*11*
	%	0.0	0.02	0.09	0.27	0.29	0.33	0.17	0.12	0.0	0.02	0.10	0.18	0.19	0.22
1	*avg*	0.0	1.45	1.27	1.31	1.29	1.30	1.31	1.32	0.0	1.38	1.30	1.27	1.33	1.28
	ω_c	0.0	0.03	0.11	0.35	0.37	0.43	0.22	0.16	0.0	0.03	0.13	0.23	0.25	0.28
	%	-	-	-	0.20	0.40	0.40	.20	0.10	0.0	0.0	0.06	0.20	0.24	0.20
2	*avg*	-	-	-	1.69	1.69	1.70	1.69	1.69	0.0	0.0	1.69	1.69	1.69	1.69
	ω_c	-	-	-	0.34	0.68	0.68	0.34	0.17	0.0	0.0	0.10	0.34	0.41	0.34
	%	-	-	-	0.24	0.60	0.16	.24	0.08	0.0	0.0	0.12	0.12	0.24	0.20
3	*avg*	-	-	-	1.69	1.70	1.72	1.71	1.70	0.0	0.0	1.70	1.70	1.69	1.69
	ω_c	-	-	-	0.41	1.02	0.27	0.41	0.13	0.0	0.0	0.20	0.20	0.41	0.34

(b) Grid search results for band count and mixture count parameters.

the same C5002 RF power generator dataset. After three iterations the ω_c measures were closely matched concluding the grid search. As the number of combinations become less, the number of top combinations used for calculating contributions was reduced from 100 to 50 and finally 25 for the last iteration. It is noted that the *detrend* method was kept into the second iteration to reaffirm its contribution. This extra precaution is because the detrend method is expensive and not preferred with respect to an embedded implementation. Upon the convergence of the grid search, it is clear that max-min local scaling, unit variance for PCA, and detrend are good choices for processing methods.



(a) Average selection criteria weight with respect to bands.



(b) Average selection criteria weight with respect to mixtures.

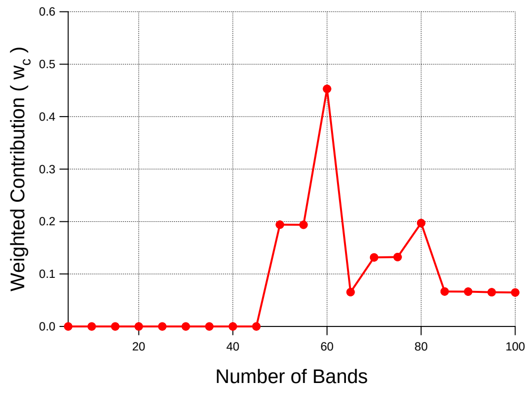
Figure 3.6: Average selection criteria weight with respect to mixture and band count for final grid search iteration. Algorithmic parameters fixed to use max-min local scaling, linear detrending, unit variance with PCA, and likelihood threshold scale of $\lambda = 3.0$.

As for zero-mean, all methods were equal contributors, therefore *no* zero-mean was selected because it has no extra computation as the detrend operation already performs a zero-mean operation. Remaining is a significant range of band and mixture counts with equal contributions. Thus a final exploration is conducted using a finer grid with methods set as aforementioned with bands counts $\{10, 15, 20, \ldots, 100\}$ and mixtures counts $\{1, 2, 3, \ldots, 15\}$. The resulting average ω_c values for each band and mixture count may be seen in Figure 3.6, where 60 bands and 8 mixtures were selected as the best parameters. Despite a single mixture having the highest contribution it was not selected because lower numbers of mixtures are known to have poor specificity (as determined from the standard dataset exploration). Also a lower number of bands, 60 instead of 80, were selected to aid in the process of PCA.

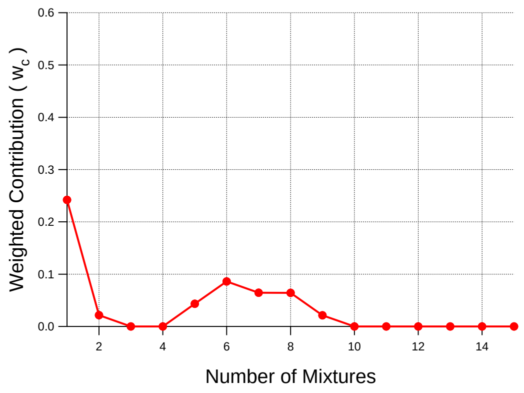
As was stated previously, the detrend operation is expensive computationally. Thus an additional grid search is conducted by repeating the 3rd iteration, as shown in Table

Table 3.13: Grid search results without detrend operation where the percent occurrence (%), average fitness (avg), and weight contribution (ω_c) for each method/parameter is reported.

	Zero Mean			**Bands**			**Mixtures**							
	N	*L*	*G*	*45*	*60*	*80*	*1*	*2*	*3*	*4*	*5*	*7*	*9*	*11*
%	0.36	0.32	0.32	0.12	0.28	0.60	0.24	0.08	0.0	0.0	0.12	0.12	0.24	0.20
avg	1.64	1.65	1.65	1.63	1.64	1.66	1.68	1.64	0.0	0.0	1.64	1.66	1.64	1.63
ω_c	0.59	0.53	0.53	0.20	0.46	0.99	0.40	0.13	0.0	0.0	0.19	0.20	0.39	0.33



(a) Average selection criteria weight with respect to bands.



(b) Average selection criteria weight with respect to mixtures.

Figure 3.7: Average selection criteria weight with respect to mixture and band count for final grid search iteration without using the detrend operation. Algorithmic parameters fixed to use max-min local scaling, unit variance with PCA, and likelihood threshold scale of $\lambda = 3.0$.

3.12, with the exception of not performing the detrend processing step. The corresponding results without detrending are reported in Table 3.13, where the methods/parameters shown are those not concretely selected. Similar methods/parameters without the detrending operation are seen, no significant zero-mean strategy, a higher mixture count (> 7), and inconclusive number of bands and mixture count. A finer grid is also run not using the detrend operation and no scaling was selected to reduce computational complexity. The results from the finer grid search without detrend can be seen in Figure 3.7. From the results, it is concluded that the classification accuracies with and without the detrend operation are closely matched.

3.3.3 Final Parameter Evaluation

The final results from the iterative grid search using the C5002 RF power generator suggests to use linear detrending and max-min local scaling for preproceesing methods, 80 bands for feature vector creation, unit variance when reducing features with PCA, and to use 6 mixtures for the GMM with 3.0 standard deviations from mean training likelihood for classification. These methods and parameters are then used to obtain GMMs and classification metrics for fingerprints from all the RF power generators. For each RF power generators, its fingerprints are partitioned as was done in the grid search, but with 50 trials instead of 25. Table 3.14 contains the average and standard deviation of the classification metrics for all the RF power generators as well as the number of principal components (PCs) used.

From the results in Table 3.14 it is apparent that the one-class classification using GMM is an effective method to assess *normal* operation of the RF power generators targeted. It is seen that by sacrificing a small amount of classification accuracy $\sim 3\%$, detrending can be avoided, the number of bands may be reduced from 80 to 60 and the number mixtures from 8 to 6. These subtle reductions may prove more significant when considering an embedded implementation.

The following is a summarizes the results of not using detrending, because they will most likely be the target for the embedded implementation. The overall average robust true positive rate is 94.76% with an average standard deviation of 2.05% yielding a 3σ value of 88.61%. The specificity values are slightly lower than the true positive rate at 77.92% and 87.51%. Theses lower accuracy values may be the result of the difficultly

Table 3.14: Classification accuracy of normal and seeded-faults for RF power generator dataset using parameters chosen from results of iterative grid search experiments with the C5002 model. General processing methods are unit variance with PCA and a threshold of 3.0 standard deviations from mean training likelihood. Results are from 50 trial runs with 15 EM generations and model selection based on validation TPR.

Generator	#PCs	V_{ACC}	V_{TPR}	V_{SPC}	R_{TPR}
LVG3527A	68.88 (2.83)	77.56 (0.74)	99.82 (1.01)	75.78 (0.79)	98.74 (4.34)
LVG3527B	231.94 (2.25)	NA	99.29 (0.35)	NA	99.09 (0.50)
LVG3560A	72.64 (2.63)	72.97 (0.00)	100.0 (0.00)	68.15 (0.00)	99.96 (0.22)
LVG3560B	81.26 (3.17)	NA	99.62 (0.60)	NA	99.31 (0.78)
C5002	71.44 (3.78)	90.15 (0.74)	99.46 (0.74)	89.31 (2.48)	97.98 (4.40)
C13002	310.08 (3.92)	NA	91.61 (1.20)	NA	91.21 (1.71)

(a) Using detrend, 80 bands, and 8 mixtures.

Generator	#PCs	V_{ACC}	V_{TPR}	V_{SPC}	R_{TPR}
LVG3527A	74.80 (1.53)	77.92 (0.33)	99.96 (0.21)	76.15 (0.34)	98.54 (1.72)
LVG3527B	261.92 (3.85)	NA	97.29 (0.64)	NA	97.07 (1.14)
LVG3560A	79.22(2.57)	66.75 (2.69)	99.95 (0.22)	66.74 (2.69)	99.87 (0.50)
LVG3560B	84.62 (4.07)	NA	97.56 (2.00)	NA	97.87 (1.65)
C5002	70.70 (4.14)	87.51 (3.85)	98.59 (1.51)	86.56 (4.11)	96.42 (4.17)
C13002	297.26 (3.70)	NA	87.90 (1.24)	NA	87.47 (1.37)

(b) Not using detrend, 60 bands, 6 mixtures.

of separating the *non-normal* fingerprints as they were seeded to mimic conditions that standard quality control methods have difficulty. Additionally, the lower specificity may be highly dependent on the limited number of *normal* samples (279 and 226) used to train and validate the GMM. Finally, a significantly higher number of principal components were required for the RF power generator models that did not have *non-normal* data. The increase in component count is most likely due to significantly larger *normal* sample sizes (2,479 and 1,545) versus (279 and 226).

A baseline comparison of the classification accuracies of the proposed method on RF power generator data is provided. The k-means algorithm was used as the baseline because of its simple classification algorithm. For the RF power generator problem there are two classes *normal* and *novel* or *non-normal*. In order to provide a comparison, data only from RF power generator with seeded faults is used. The k-means classification algorithm requires data from all classes during training for learning means or centroids for each class. Seeded fault data was shuffled and split 50/50 then placed respectively in the training and validation data sets. Robust data was removed similar to previous tests with data from 30% unique units. The robust data only contains normal data to simulate a real-world case where only normal data is known. The remaining normal data was split 50/50 into training and validation sets. The standard k-means implementation is used with random initialization and distance measured using the squared euclidean distance. The k-means algorithm was applied to the output of the reduced feature vector training set (result after PCA) to replace EM and Gaussian mixture models. The centroids produced from training via k-means were used to evaluate the classification of validation and robust data. Classification for each feature vector is decided as the closest centroid.

Table 3.15: Comparative results of k-means and GMM methods on classification of RF power generator data. Results are from 50 trial runs and 15 model generations per trial and selection based on TPR.

Metric	**Method**	**LVG3527A**	**LVG3560A**	**C5002**
$\#PC$	k-means	17.42 (35.77)	39.14 (8.16)	12.10 (4.21)
	GMM	74.80 (1.53)	79.22 (2.57)	70.70 (4.14)
V_{ACC}	k-means	54.56 (0.02)	60.06 (0.18)	57.86 (0.27)
	GMM	77.92 (0.33)	72.97 (0.00)	90.15 (0.74)
V_{TPR}	k-means	99.98(0.16)	100.00(0.00)	100.00(0.00)
	GMM	99.96 (0.21)	99.95 (0.22)	98.59 (1.51)
V_{SPC}	k-means	10.03 (0.05)	25.18 (0.56)	18.64 (0.76)
	GMM	76.15 (0.34)	66.74 (2.69)	86.56 (4.11)
R_{TPR}	k-means	99.98 (0.16)	100.00(0.00)	100.00(0.00)
	GMM	98.54 (1.72)	99.87 (0.50)	96.42 (4.17)

Table 3.15 provides the comparative results of the proposed GMM one-class classifier to k-means. The preprocessing methods used were local zero-mean, local max-min scaling, no detrend, unit variance PCA with 99% variance, and 80 bands. 50 trial runs are used with the data shuffled each trial. Additionally, for each trial 15 generations of models (GMM or k-means) are used to reduce the effect of randomness in model initialization. From Table 3.15 it is seen that the GMM model approach is far superior in specificity with respect to k-means. In all, results conclude that one-class classification using GMM can successfully identify with reasonable accuracy *normal* versus *non-normal* system operation of an RF power generator.

3.4 Embedded Implementation

The proposed method was originally implemented using MATLAB computing software because of its ability for rapid algorithm development and extensive toolboxes and libraries. However, the end goal is to implement the ONDS within the target embedded system. C++ was chosen as the language for the embedded implementation for its speed, multi-platform support, and object-oriented capabilities. Additionally, no 3rd-parties libraries are used in order to avoid platform dependencies with additional libraries. Moreover the C++98 standard was selected for backward compatibility with legacy systems with compilers that do not support C++0x standards.

Without 3rd party libraries custom math classes and functions were implemented. Among the custom math classes were matrices and vectors for real and complex numbers. Mathematical operators were implemented/overloaded for matrix multiplication, scalar multiplication/division, element-wise multiplication/division/addition/subtraction, and transpose. Some additional operations implemented are n-dimensional sum, mean, max, min, variance, and co-variance. All operations and operators were implemented to conform to MATLAB syntax and validated for output.

For testing of the embedded implementation of ONDS the chosen platform was a BeagleBone Black (BBB) with a 1-GHz Sitara$^{\text{TM}}$ARM®Cortex-A8 running a 32-bit Ubuntu 13.04 distribution. The OS was set to run on the BBB's embedded memory (eMMC) which offers a total of 2GB of space. The embedded code developed in this work was compiled with g++ 4.7.3 with C++98 standard and no extra optimization. Since the BBB is not one of the RF generators, fingerprint data files were uploaded to the BBB to simulate data collection. The one-class classifier read the data files then performed the algorithm and output to console classification of the file(s) as normal or not-normal. The resulting implementation classification output matched that of the MATLAB version.

Within an embedded environment importance is placed on code footprint (static memory required), dynamic memory, and speed of execution. Table 3.16 is a summary of recorded memory footprints and execution time. Execution time is the total time to

Table 3.16: Embedded implementation's code footprint, memory usage and algorithm execution time.

Total Execution Time	600ms - 950ms
Model Information Footprint	113KB
Source Code Footprint	60KB
Executable Footprint	76.8KB
Total Code Footprint	250KB
Dynamic Memory Usage	1.1MB

classify a fingerprint, which includes time to read fingerprint data from a file as well as model information. The dynamic memory was determined by monitoring process memory consumption and recording the maximum memory consumed during execution. Reported measures are based on the model using a single Gaussian component ($k = 1$). Based on the memory analysis there exists no physical constraints for the code to be ported over to an RF power generator.

Chapter 4

Equine Distress

Presented in this chapter is an application of the ONDS to animal health monitoring, where the data is random in nature. Equine distress can present in many ways and stem from a multitude of factors including injury, trauma, illness, fear, and boredom. There are a number of common acute equine distress conditions, including colic and casting that can negatively impact a horse and result in serious injury or death if intervention is delayed. Colic is the leading natural cause of death in horses other than old age, and the American Association of Equine Practitioners (AAEP) estimates more than 900,000 horses in the United States will experience an episode of colic this year [71, 72]. Despite this large incidence, $\sim$90% of these cases can be treated medically and without the need of invasive emergency surgery if detected early [73, 74]. Colic is a symptom of disease, but not a disease itself, and is generally defined as any abdominal pain. Equine colic can involve any number of abdominal organs, not just the gastrointestinal tract. For example, abdominal discomfort from kidney or liver disease will sometimes cause signs of colic. Equine colic can originate from the stomach, small intestine, large intestine, or some combination thereof, and is associated with any malfunction, displacement, twisting, swelling, infection, or lesion of any part of the equine digestive system. A horse suffering from colic may show any number of signs [72]:

- Pawing and/or scraping (front legs)
- Kicking (back legs) up, or at abdomen
- Repeated lying down and rising/standing
- Rolling (+/- thrashing)
- Stretching
- Pacing
- Flank watching (i.e., turning of the head to watch stomach and/or hind quarters)
- Biting/nipping the stomach

Horses with a history of colic or who have had prior abdominal surgery are at 3-5 times greater risk for further episodes of colic [75]. There are also behavioral risk factors that correlate with increased risk of recurrent colic within 12 months of a colic event, including horses that either crib/windsuck or weave. Horses that crib or windsuck have more than a 10-fold chance of experiencing recurrent colic, and horses that weave have nearly a 4-fold chance [76].

Unfortunately, the detection of colic can be difficult because it is multifaceted and often occurs overnight or at remote locations when and where human caretakers are not present. Even when caretakers are present, diagnosis can be elusive with symptoms ranging from subjective and subtle changes in the animal's attitude (e.g., depression) to objective changes in vital signs (e.g., increased heart and respiratory rates, rise in temperature), biologic functions (e.g., lack of digestion), and behavior (i.e., repeated characteristic motion patterns of pawing, kicking, flank watching, rising/falling, rolling with/without thrashing, lack of healthy shaking, etc). Prior information of a horse's normal behavior can significantly assist identification of any deviation from normal that could correlate with distress, pain, or other aspects of discomfort. Current diagnostic methods used by horse owners and caretakers rely on *subjective* behavioral observation for understanding of a horse's normal behavior and detection of colic. Subjective and intermittent observation of motions, activities, and posture are potentially inefficient with respect to consistent identification of *non-normal* or *novel* behaviors that may correlate with distress such as colic. Thus, a remote wearable monitoring device, non-invasively attached to a horse, that can acquire and analyze real-time data of their behavior (i.e., motions), and alert caretakers for early intervention at the first signs of novel events may lead to improved survival and the quality of life outcomes for many horses.

Recent advances in computational power available for small wearable devices have allowed for higher complexity of on-device algorithms. Thus, machine learning algorithms and other signal processing methods may be implemented to achieve remote wearable monitoring devices that can acquire and analyze real-time data. Additionally, improvements in motion sensor quality and availability have provided access to significant quantity/quality of motion data for statistical analysis and modeling. Therefore, the proposed ONDS is an excellant candidate for identification of equine distress. Following

4.1 ONDS For Equine Distress

Novel events that correspond to novel behavior are windows of motion data that are unlikely to belong to the learned model of normal. To illustrate a novel event for equine behavior, a *roll* behavior is shown in Figure 4.1. A roll behavior, or when a horse is rolling on the ground, is typically an infrequent behavior. Furthermore, a roll that has frequent repetitious occurrences could be an indication for colic and thus is a good target as a novel event. Figure 4.1 contains a single roll behavior and represents an ideal novel event with respect to its corresponding raw sensor values. The raw sensor values, in Figure 4.1, depict that the signals from a roll behavior contain high-frequency components that are not present in surrounding sensor values. Thus, it is inferred that good features to use for modeling novel events are frequency components. The next section explains how the raw signals are collected and processed prior to frequency analysis.

4.1.1 Data Collection and Preprocessing

The raw data is provided by a 9-axis IMU, and is further comprised of a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. Each of the sub-sensors of the IMU has their own maximum sampling rates, thus the data collection embedded software obtains sensor values asynchronously as they are reported by the individual sensors. The asynchronously reported values are then stored/sampled at a chosen fixed rate. To provide continual evaluation of motion data, the data collection process is implemented using a moving window. A moving window of raw sensor data is collected for a specified window length and window overlap. The window length, N, is selected based on the sampling

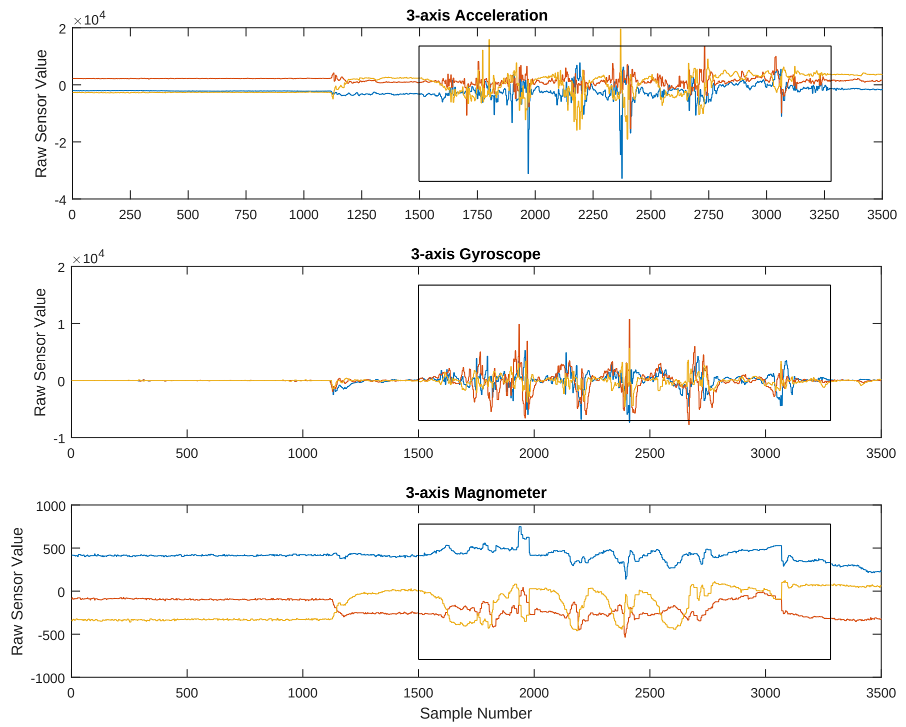


Figure 4.1: Example hand-segmented novel event of a roll behavior.

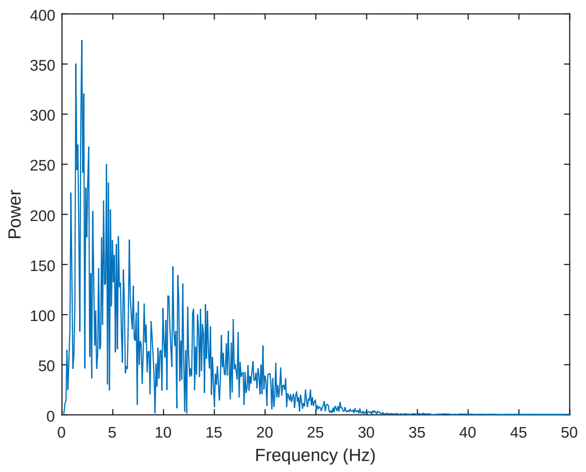
frequency of raw data (F_s) and the expected period of time for a suspect novel event (T_s). The sampling frequency, F_s, is selected to be 100 Hz and T_s is chosen to be 10 s based on observation of equine behaviors. Therefore, $N = 1000$, but for computational efficiency in frequency analysis the next power of 2 is used, yielding $N = 1024$. A window of data is collected by moving the previous window by the window overlap (Δ). The selection of Δ is chosen to be the shortest period of time for a suspect novel event. This time was observed to be ~ 2.5 s, yielding $\Delta = 256$.

The raw data provided use 16-bit signed numbers and can allow for a range of values in [-32768, 32767]. However, not all the sensors use the full range and some contain non-zero offsets. Therefore, data standardization is performed to improve multivariate analysis. The z-score standardization method is used such that all the variables have equal means of 0 and standard deviations of 1.

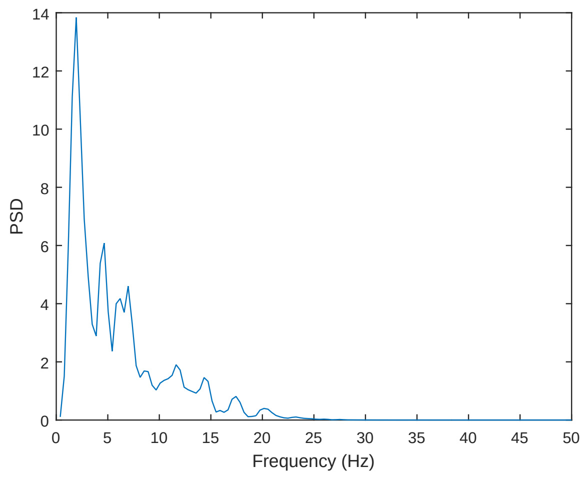
After standardization, the raw motion data are still very noisy due to the nature of the data collection and the sensors themselves. To improve the significance of frequency analysis, the raw signals are filtered to remove noise. There is a reasonable assumption that an animal is incapable of producing natural motion beyond a particular frequency range. Thus, an animal's general motion is typically very low-frequency; concluding that high frequencies present in the raw data are most likely the result of sensor noise. To remove the signal noise, a low-pass and a high-pass filter are used on each window of data. The Butterworth filter design is chosen for both the low-pass and high-pass filters because of its flat response in the passband and adequate filter response with minimal number of coefficients [77]. The additional high-pass filter is used to remove any large impulse that occurs around 0 Hz. The Butterworth low-pass filter was designed for a cutoff frequency of 20 Hz with three coefficients, whereas the high-pass filter was designed with a cutoff frequency of 0.5 Hz and 3 coefficients. After filtering, the feature vector creation procedure may proceed.

4.1.2 Feature Vector Creation

A novel event for this application of equine behavior is a window of motion data that contains frequency components that are typically not generated during normal activity. These frequency components can be calculated by using Fourier analysis. The equine motion data from the sensors is random with limited periodicity, thus an estimate of the PSD is instead used of the power of FFT coefficients. The specific PSD estimate used is Welch's method of periodogram averaging [53]. A visual difference in the FFT and



(a) Power using absolute value of FFT.



(b) PSD using Welch's method.

Figure 4.2: Single-sided power spectrum using FFT compared to the estimate of PSD using Welch's method.

Welch's estimate of PSD for a window of motion data with a roll behavior for a single variable is shown in Figure 4.2. From Figure 4.2, it is apparent that Welch's estimate of PSD provides more distinct frequency components in comparison to the power of the FFT coefficients. Furthermore, the sectioning during Welch's method significantly reduces the number of coefficients. In Figure 4.2, the FFT has 513 coefficients and the PSD has 129. The PSD is calculated for each individual window of data and for all nine variables. After estimating the PSD of a window of motion data, the result is a data structure that has dimensions $[N_C \times M]$, where N_C is the number of PSD coefficients and M is the number of variables.

The actual frequencies within the motion data are subject to variability due to the randomness of the signal, thus neighboring PSD values are banded together. The band-power of the PSD sums the power estimates within a predetermined number of bands of frequencies. The number of bands was selected such that each band spans a range of ~ 1

Hz. The actual number of elements in each band is calculated by using Equation (4.1), where B_f is the desired frequency range of each band, N is the number of samples in the window, F_s is the sampling frequency, and Δ_B is the resulting number of elements in each band. In the case of uneven division of bands, the highest frequencies are combined as they are expected to be close to zero.

$$\Delta_B = \left\lfloor B_f \left(\frac{N_C}{F_s \;/\; 2} \right) \right\rfloor \tag{4.1}$$

Using $B_f = 1.0$, $N_C = 129$, and $F_s = 100$, the number of elements per band is 2. The actual number of bands used after bandpower can be calculated by using Equation (4.2). Using the values previously defined, the number bands (N_B) is 65.

$$N_B = \left\lfloor \frac{N_C}{\Delta_B} \right\rfloor + 1 \tag{4.2}$$

When reshaped into a single feature vector, the feature vector's dimensionality becomes $[1 \times (N_B \cdot M)]$ and for S window samples, the data becomes $[S \times (N_B \cdot M)]$. This high dimensionality ($(N_B \cdot M) = 65 \cdot 9 = 585$) can cause computational challenges with respect to model learning. Therfore, PCA is used to reduce the number of features. Based on experimental results, the typical number of PCs kept are ~ 125 when keeping $p\% = 99.00$ of the variance. Thus, a $\sim 80\%$ reduction in feature size for $N_B = 65$ and $M = 9$ may be achieved. After feature reduction, the data is in an appropriate format for a model to be learned.

4.2 Experimental Results

The use of one-class classification techniques and GMMs is applied to novel event detection of equine behavior and validated via experimental analysis of offline data. The one-class classier and GMMs are applied to a special equine behavior dataset collected specifically for the target application. The novel event detection is explored using various modeling parameters and performance quantified using common performance metrics and expertly extracted novel events. Prior to the application of the proposed approach to novel event detection, its performance was compared to other datasets and classifiers.

4.2.1 Experimental Setup

The effectiveness of the novel event detection and selection of algorithmic parameters is based on a dataset of equine behavior from five different horses or *units*. Each unit was equipped with battery-powered data collection hardware and data sampled from a 9-axis IMU. The data collection was performed during the evening hours (6:00PM - 6:00AM) and each unit was in stall during these times. This time was selected as it is an appropriate period of time where additional non-invasive behavioral monitoring is typically needed. Each stall was supplied a camera to record the movement of the units during data collection. The cameras and data collection hardware were synced using a common Network Time Protocol (NTP) server to correlate time-stamped data samples to specific video frames. This correlation is essential for validation of the performance of novel event detection. Specifically, the dataset is a collection of asynchronous time-stamped IMU data samples that are sampled at 100 Hz and stored in Comma Separated

Table 4.1: Summary of test equine motion dataset. Each file contains 1024 data samples of 9-axis IMU data, windows are created using a moving window of size 1024 with window increment of 256, and total time is based on synchronized system clock times.

Unit	**File Count**	**Window Count**	**Time (minutes)**	**Time (hours)**
A	10304	41213	2078	34.63
B	11453	45809	2312	38.53
C	11242	44965	2270	39.50
D	11363	45449	2293	38.22
E	9620	38477	1940	32.33

Value (CSV) files with 1024 data samples per file. This equine motion dataset spans three consecutive days and its raw data size is summarized in Table 4.1.

Due to the potential open-ended combinations of behaviors that could be considered novel, a small set of behaviors were selected to evaluate the performance of the novel event detection algorithm. The behaviors selected are briefly described by the following list.

- **Fall** - Moving from standing to a laying position.
- **Roll** - Turning over on the ground.
- **Rise** - Moving from laying to a standing position.
- **Shake** - Shaking head/body after rising from the ground.

Videos from all the units are reviewed and event timestamps extracted for the target behaviors. The total number of target events is tabulated in Table 4.2. To quantify the performance of the novel event detection algorithm, the Negative Predictive Value (NPV) binary classification metric is used. Using the data collected using the experiments discussed in this subsection, we explore the effect of the various algorithmic parameters of the novel event detection algorithm.

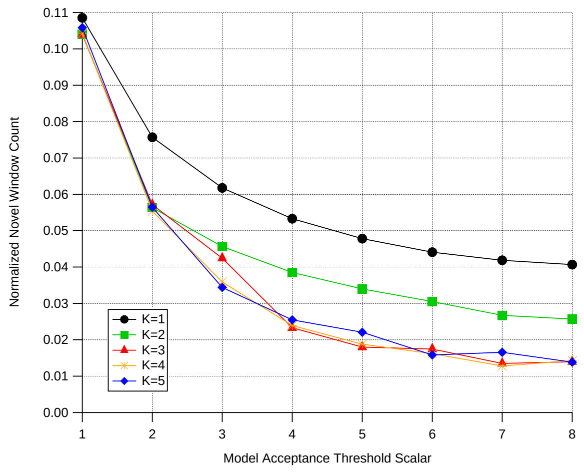
Table 4.2: Summary of target behavior event count for all units.

Unit	**Fall**	**Roll**	**Rise**	**Shake**	**Total**
A	15	10	15	11	51
B	13	2	13	8	36
C	12	5	12	4	37
D	15	5	15	3	38
E	11	1	11	1	24

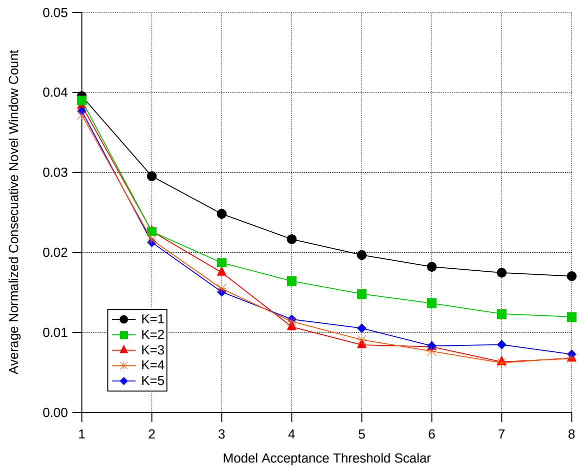
4.2.2 Novel Event Exploration Results

As discussed in Section 4.1.1, the data was collected using a moving window with size 1024 and increment of 256. The windows are standardized using the z-score, and filtered using a low pass then high pass filter with cutoff frequencies of 20 Hz and 0.5 Hz. Feature vectors for each window are created from bandpowers of the power spectrum density from each variable, estimated using Welch's method. Feature vectors are created for all the data collected in the equine behavior dataset. Algorithmic exploration is provided for different model acceptance threshold scalars (λ) and different number of mixture components (K). This exploration provides insight into the effect of thresholding and mixture components with respect to the number of novel events detected. Gaussian mixture models are learned on a per unit basis where data partitioning includes training sets and a validation sets with ratios of 50%. Principal component analysis is performed on the training sets to find projection matrices that are then used to reduce all the feature vectors. The reduced feature vectors from the training sets are used to learn a GMM of normal behavior for each unit. The EM algorithm is performed 25 times per training set and the model with the best TPR.

To evaluate the effect of the model acceptance threshold scalar, (λ), and mixture component count (K), various values are considered. For each threshold level and mixture



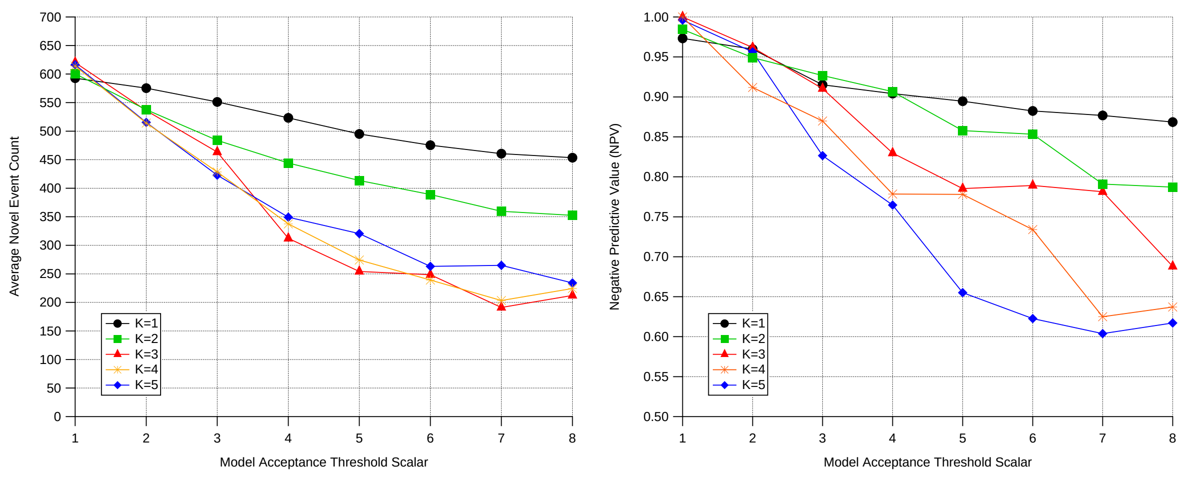
(a) Novel Windows



(b) Merged Consecutive Novel Windows

Figure 4.3: Average novel window counts normalized to total window count per unit with respect to model acceptance values for different number of Gaussian mixtures. The merged consecutive novel windows are novel windows that are consecutively classified as novel and are counted only once.

count, multiple performance metrics are averaged across all units and reported. These performance metrics include: novel window count, time per event, total time, and NPV. Figure 4.3 shows normalized novel window counts and merged window counts. The window counts are normalized with respect to the total number of windows collected for each unit. To reduce the number of novel windows, windows that are consecutively classified as novel are merged into one larger window as in Figure 4.3b. From Figure 4.3, it is seen that values of $\lambda > 6$ do not significantly further reduce novel event count and mixture component counts $K = \{3, 4, 5\}$ produce less novel windows. Also, prior to merging novel windows less than 10% of all the windows are considered novel indicating that the GMMs are effectively estimating normal behavior on a per unit basis. The effect of thresholding and mixture component count on novel event count and NPV is provided in Figure 4.4. In Figure 4.4a, a novel event is a segment in time with novel windows that are within 30 s of each other. The NPV values were calculated based on

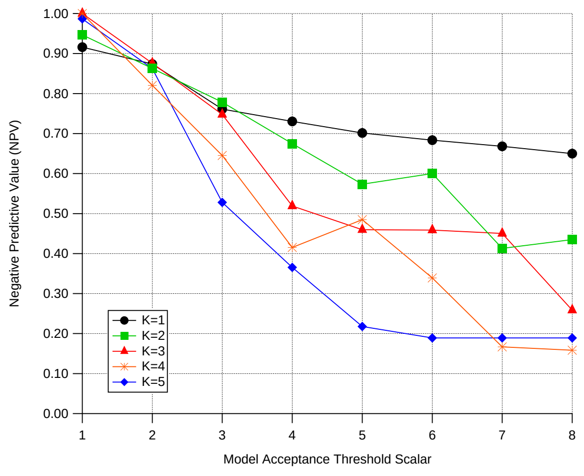


(a) Novel Event Count (b) Negative Predictive Value (NPV)

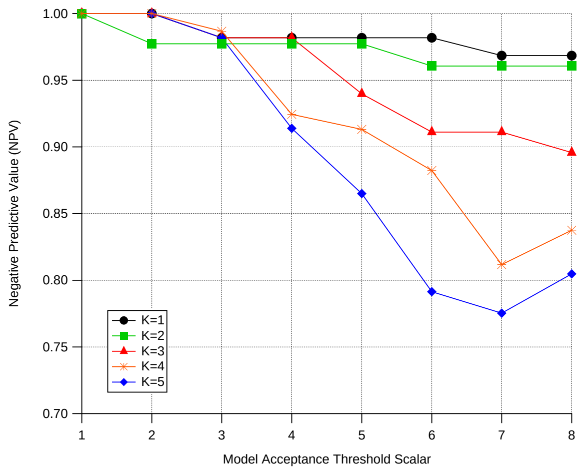
Figure 4.4: Average novel event count and negative predictive values with respect to model acceptance threshold values for a different number of Gaussian mixtures.

the hand targeted events from the fall, roll, rise, and shake behaviors. Figure 4.4 is a clear side-by-side comparison of the trade-off between number of novel events versus NPV. Using a lower threshold and lower mixture count produces higher NPVs at the cost of significantly more novel events. High novel event count suggest an increase in *false positives* where events are classified as novel when in truth they were normal. Moreover, increased numbers of novel events require more processing power to validate them and if used in an embedded application it may consume too much processing power. Figure 4.5, provides the NPV for the fall and rise behaviors alone. The roll and shake behaviors were not reported as they are nearly always detected as novel event due to the magnitude of their high frequency components. The fall behavior exhibits the worst NPV and is most likely due to the less dramatic nature of the fall motion profile and short event time. To provide a better understanding of the amount of motion data that is classified as novel, the amount of time per event and total event time is reported in Figure 4.6.

Using the data collected and metrics reported in Figures 4.4-4.6, some final parameter

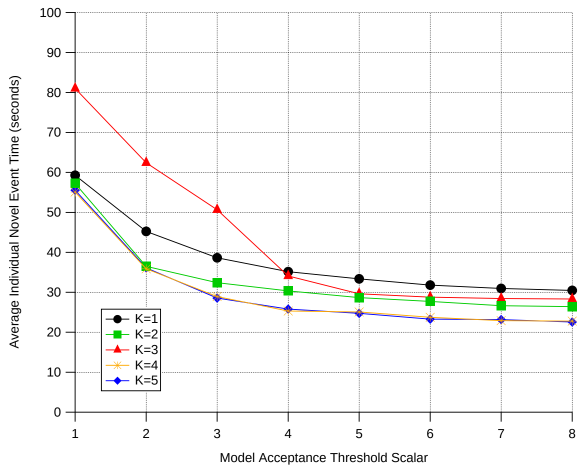


(a) Fall NPV

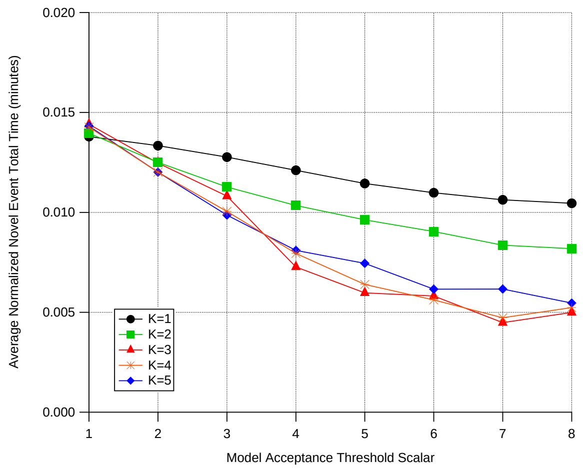


(b) Rise NPV

Figure 4.5: Average negative predictive values (NPV) for fall and rise behaviors for different model acceptance threshold values for different number of Gaussian mixtures.



(a) Time per Event



(b) Normalized Total Time

Figure 4.6: Average time per novel event in seconds and total time normalized to the total collection time with respect to model acceptance threshold values for different number of Gaussian mixtures.

selections of threshold and mixture count is concluded. Based on prior knowledge through observation of novel equine behaviors, the behaviors typically span from 3-30 s per event. This span indicated that a threshold scalar (λ) between 3-6 is appropriate as supported by Figure 4.6a. Additionally, Figure 4.3 shows that mixture counts of $K = \{3, 4, 5\}$ demonstrate similar novel window count and are less than $K = \{1, 2\}$ mixtures. Figure 4.4b suggests that $\lambda = 3$ mixtures provides the best NPV compared to $\lambda = \{4, 5\}$. Additional experimental runs for robust analysis is done to further explore the effects of λ and K.

4.2.3 Individual Model Results

Based on the exploration grid of thresholding (λ) and mixture count (K) (previous subsection), a mixture count of $K = 3$ and a threshold level of ($\lambda = 3$) are selected for further experimental runs. The threshold scalar value of ($\lambda = 3$) is chosen for balance of higher NPV and lower novel event count. Tables 4.3 and 4.4 summarize the performance metrics for each individual unit using $\lambda = 3$ and $K = 3$.

From Table 4.4, it is seen that unit A demonstrates the lowest NPV values. Referring back to Table 4.1, unit A was also the unit with the most target behaviors, indicating the horse associated with unit A is potentially more active. More active units may require additional data samples for training to better estimate normal behavior. The training of behavior models from independent horses requires extensive data collection, storage, and processing using high-performance machines. Therefore, it is advantageous to explore the possibility of a single generic model that can provide comparable novel event detection

Table 4.3: Novel event count per unit averaged over 50 trial runs with 25 EM each run using a model acceptance threshold scalar $\lambda = 3$ and mixture count $K = 3$. Standard deviations of numbers reported are provided in parenthesis.

Unit	**Window Count**		**Event Count**	**Novel Time**	
	Novel	*Consecutive*		*Per Event (sec)*	*Total (min)*
A	2363.20 (91.33)	797.24 (47.77)	495.10 (14.01)	31.76 (9.55)	262.11 (14.81)
B	2030.20 (54.36)	980.98 (24.78)	554.22 (13.12)	35.40 (0.55)	327.01 (8.49)
C	1167.20 (114.08)	538.02 (40.57)	351.62 (16.11)	28.15 (1.21)	165.22 (13.89)
D	1395.40 (105.98)	601.70 (35.83)	383.44 (18.32)	29.48 (0.83)	188.56 (12.68)
E	1765.70 (129.80)	737.02 (48.41)	478.68 (34.68)	30.04 (1.34)	239.33 (16.90)
Avg.	**1744.34 (99.11)**	**730.99 (39.47)**	**461.32 (24.06)**	**30.97 (2.70)**	**236.45 (13.35)**

Table 4.4: Negative predictive values (NPV) per unit averaged over 50 trial runs with 25 EM each run and a model acceptance threshold scalar $\lambda = 3$ and mixture count $K = 3$. Standard deviations of numbers reported are provided in parenthesis.

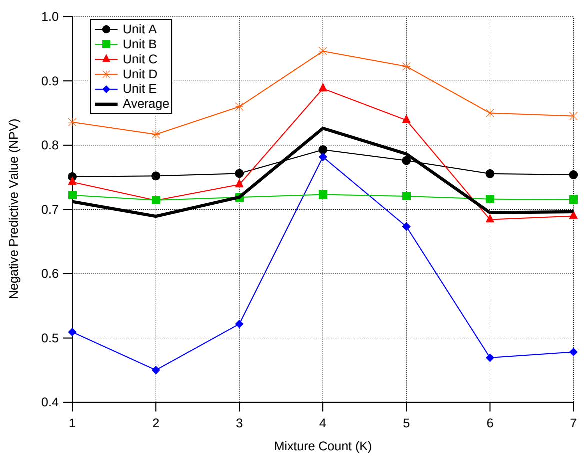
Unit	**NPV**	**Ratio Behaviors Detected as Novel**			
		Fall	*Roll*	*Rise*	*Shake*
A	0.7714 (0.0212)	0.2440 (0.0532)	1.0000 (0.0000)	0.9787 (0.0314)	1.0000 (0.0000)
B	0.9256 (0.0142)	0.7939 (0.0394)	1.0000 (0.0000)	1.0000 (0.0000)	1.0000 (0.0000)
C	0.8562 (0.0468)	0.5908 (0.1332)	1.0000 (0.0000)	1.0000 (0.0000)	1.0000 (0.0000)
D	0.9732 (0.0037)	0.9320 (0.0094)	1.0000 (0.0000)	1.0000 (0.0000)	1.0000 (0.0000)
E	0.9858 (0.0200)	0.9320 (0.0094)	1.0000 (0.0000)	0.9691 (0.0435)	1.0000 (0.0000)
Avg.	**0.9024 (0.0212)**	**0.6985 (0.0489)**	**1.0000 (0.0000)**	**0.9896 (0.0150)**	**1.0000 (0.0000)**

performance.

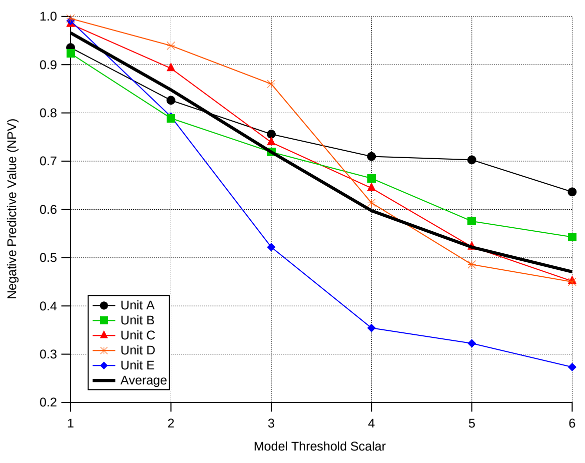
4.2.4 Generic Model Results

To assess the performance of a generic equine behavior model, experimental data was analyzed similar to the individual models. A generic model was created across behavioral data from multiple units. For the generic model learning, the data was partitioned such that 3 of the 5 unit's data was used for training and validation. The remaining 2 units are isolated for robustness test. From the training and validation data, randomized feature vectors are split into 50% for training and 50% for validation. Using the training data, 25 generic models are learned using EM. Using the validation data, the best generic GMM is selected as the model that has the highest TPR. Performance metrics of novel window count and NPV are recorded for all data including training, validation, and robust. This entire process is repeated 50 times with the data partitioning randomized for each trial. Figure 4.7a shows the average NPV values across all trials for all units using $K = \{1, 2, 3, 4, 5\}$ and $\lambda = 3.0$. The value of λ was selected based on the results of the individual models. The peak in NPV (Figure 4.7a) suggests the best mixture count for generic models is $K = 4$, where the average NPV is 82.64%. Figure 4.7b uses a mixture count $K = 4$ and shows NPV with respect to model threshold values of $\lambda = \{1, 2, 3, 4, 5, 6\}$. From Figure 4.8a, it is seen that the generic model NPVs trend similarly to the individual models.

Figure 4.8 is provided for comparative analysis of generic and individual behavior models. From Figure 4.8, it is seen that the individual behavior models are able to



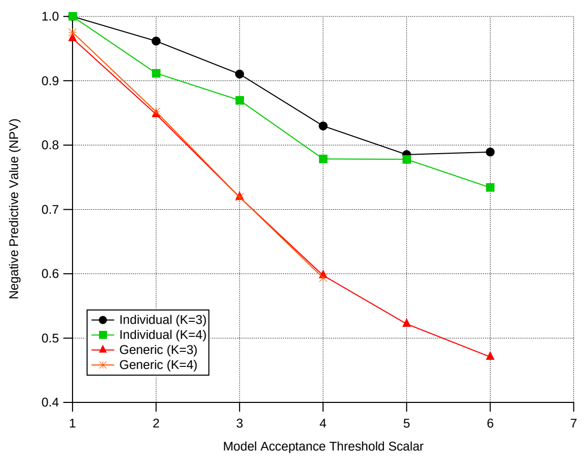
(a) NPV vs Mixture Count



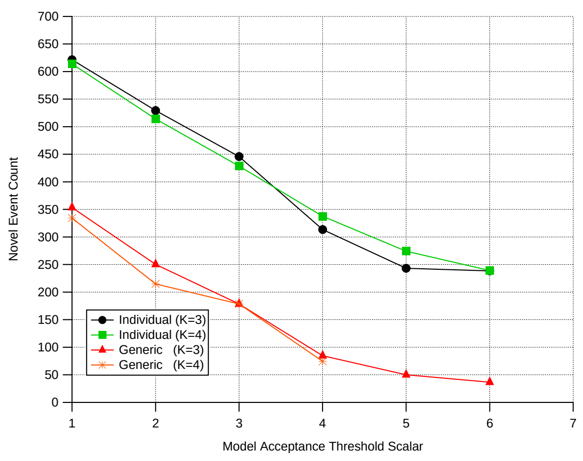
(b) NPV vs Threshold

Figure 4.7: Generic model analysis presented as average values of 50 trials of negative predictive values with respect to mixture count and thresholding. For (a) each model's threshold value was set to be $\lambda = 3.0$ and for (b) the number of mixtures was set to $K = 4$.

achieve higher NPV values compared to the same model acceptance scalar (λ) used with generic models. However, Figure 4.8b shows that by using generic behavior models, the number novel event is significantly reduced. Therefore, using a generic model with $\lambda = 1.0$ and $K = 4$, an average NPV value of 97.5% is achievable as opposed to 77.9% for individual models with a comparable novel event count ($\lambda = 4.0$). These values may converge with additional data per unit, because the generic model training uses ~ 3 times more data. Nevertheless, the significant difference between generic and independent models demonstrate the strength of mixture models. An accurate single generic model of behavior can be learned and applied to multiple units successfully; even those units not used in training or validation (robust units). Regardless of the use of generic or individual models, the novel event detection algorithm implementation as an embedded solution is required. Hence, experimentation with respect to computational complexity via execution time is provided. In the next section, we test an implementation of the



(a) NPV vs Threshold



(b) Event Count vs Threshold

Figure 4.8: Performance of generic behavior models versus individual behavior models with respect to NPV and novel event count. Values provided are average values across all units with 50 trials performed per unit.

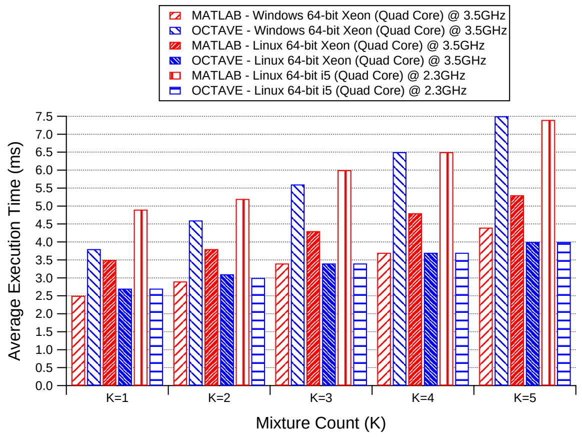
novel event detection algorithm within an embedded system.

4.2.5 Embedded Implementation

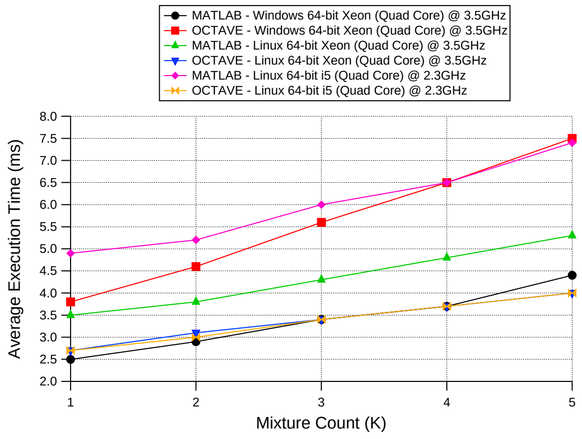
The target application of the novel event detection is an algorithm embedded in a wearable device. In order for the novel event detection algorithm to be embedded on-device it must have a computational complexity suitable for current microprocessors. To evaluate computational complexity, execution time of processing a single window of data for various platforms is considered. For embedded testing, a GMM model was learned for a specific unit (Unit E) and the model parameters stored in a file. Two windows of data are pre-collected to represent a normal event and a novel event. These windows are also stored in a file to be imported on a target platform. For each platform explored, the model information and windows of data are loaded into memory. Each window is then subjected to novel window classification, which is comprised of: preprocessing, feature vector creation, banding, feature reduction, and one-class classification. The resulting binary classification and log-likelihood values are compared to the original values for validation of computational accuracy. The average of the overall execution time of the novel window classification of the two windows is recorded. Additionally, to analyze the effect of number of mixtures (K), average execution times are reported for GMM with $K = \{1, 2, 3, 4, 5\}$.

The original novel event detection algorithm is written using the MATLAB® computing language [78]. However, with a target application of an embedded device, the MATLAB run-time environment is not practical because of its size and dependency on the x86 instruction set. Therefore, the GNU Octave language and its Command Line Interface (CLI) is selected for comparison to MATLAB. Octave is selected because of its ease of portability from MATLAB and support of the ARM® instruction set [79]. Furthermore, most embedded applications use a Linux-based operating system (OS) as opposed to Microsoft Windows. Therefore, computational time using MATLAB and Octave are explored for both Windows and Linux operating systems. It is noted that regardless of platform or operating system, the calculation of log-likelihood values are found to be accurate to at least four significant digits.

Figure 4.9 tabulates the execution time of 500 trials of novel window classifications on high-performance CPUs for Windows and Linux operating systems. From Figure 4.9, the difference between execution times of Octave and MATLAB are apparent, but it can be mitigated by selecting the proper language for the targeted operating system. From the experiments, MATLAB consistently demonstrated lower execution times on Windows,



(a) Bar chart representation.

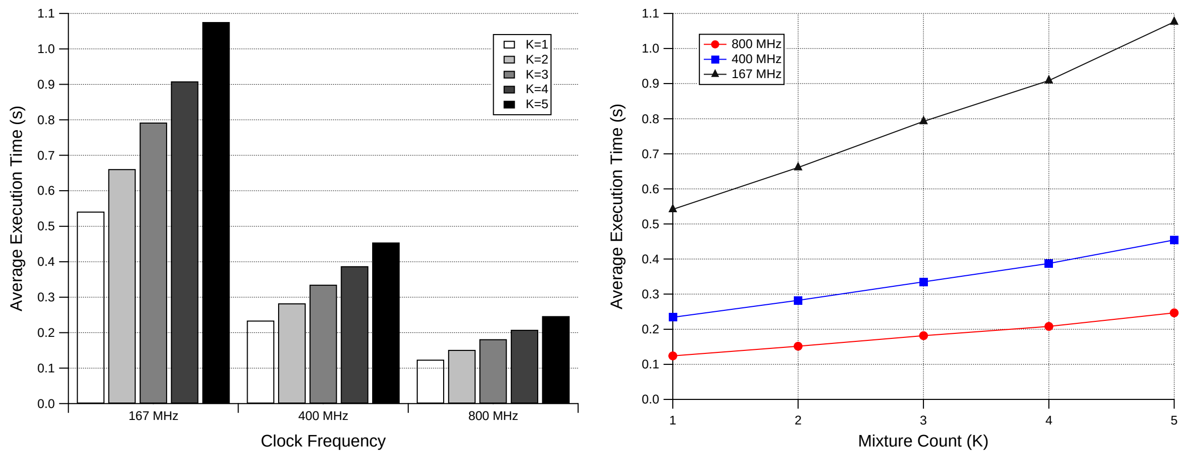


(b) Line graph representation.

Figure 4.9: Average execution time of 500 trials of novel window classification on different high-performance CPUs with Linux and Microsoft Windows operating systems.

whereas Octave reported lower execution times on Linux. The platform dependent performance is most likely due to the tool-sets used to compile each language (e.g. the GNU tool-sets are generally more optimized for development in Linux) and/or software drivers for hardware. However, the end-application targets the Linux operating system and an Octave solution is shown (Figure 4.9) to be closely matched in execution time compared to a MATLAB solution on Windows. The Octave/MATLAB execution times on proper operating systems with the same hardware recorded less than 1 ms differences.

A embedded platform architecture using Linux and Octave is selected for testing based on the confirmation of accuracy and comparable execution times. Current embedded/wearable devices are using ARM-based microprocessors due to their small footprint, high computational speeds (near 1 GHz), and low-power consumption. For algorithm testing and execution time exploration, the ARM® Cortex™-A8 or more specifically the Freescale i.MX53 is selected. The Freescale i.MX53 boasts a 32-bit 800 MHz processor and the development board used has 512 DDR3 RAM and runs a Arch Linux operating



(a) Bar chart representation (b) Line graph representation

Figure 4.10: Average execution time of 500 trials of novel window classification on the ARM® Cortex™-A8 with Linux operating system.

system from an external flash memory card. The Freescale i.MX53 can be scaled to operate at clock frequencies of 800 MHz, 400 MHz, and 167 MHz. Each of these frequencies are used to evaluate the execution time of novel window classification for GMMs with mixture count $K = \{1, 2, 3, 4, 5\}$. From Figure 4.10, the effect of clock frequency is seen. Using a mixture count of $K = 4$, as selected from the results in Section 4.2.3, and a clock frequency of 800 MHz, the execution time is ~ 0.2 s. Also, from Figure 4.10 a very subtle exponential growth in execution time is seen as the number of mixtures increase, which is more prominent with lower clock frequencies. A single window has 1024 samples that are sampled at 100 Hz yielding a single window collection time to be 10.24 s. However, with a moving window every 256 samples or 2.56 s, a window should be classified within this time. Even at 167 MHz, the novel window classification is only 0.8 s, and thus suitable for use in an embedded application. Moreover, the execution times presented are for processing an entire window and it is noted that further speed-up can be achieved through optimization based on the moving window scheme. Finally, the execution times reported do not consider the additional background processing that may be required such as data collection, windowing, and other extra required background processes. These other processes were not considered to provide a raw evaluation of execution time of the novel window processing on a embedded platform with an ARM-based processor using Octave in Linux.

Chapter 5

Conclusion

Presented is an in-depth analysis of the Online Novelty Detection System (ONDS) as a one-class classifier implemented using Gaussian mixture models. Specific applications to identifying systemic health of RF power generators and novel behavior in equine motion data have been implemented successfully. The data modeled and classified are multivariate time-series datasets from sensors with a wide range of values and under different operational conditions. Generic functional blocks have been defined as data collection, preprocessing, feature vector creation, model learning/testing, and classification. Many of these procedures are common in literature. Typically, focus is placed on the model learning procedure and classifier tuning. Our approach differs with a heavy focus on exploring different preprocessing methods and selecting optimum algorithm parameters. Using a thorough and methodical exploration, preprocessing methods and parameters are experimentally selected.

With respect to the application using RF power generator, high overall average robust classification accuracy (94.76%) with low deviation (2.05%) of *normal* operation of the RF power generators were achieved. The average specificity values of 87.51% and 77.92% were achieved and noted to be significant considering the *non-normal* conditions tested were actual *non-normal* fingerprints from seeded faulty conditions. Additionally, these values reflect the limited *normal* samples available for learning an adequate representation of *normal* for these units. Furthermore, the *normal* fingerprint data collection is typically done prior to the unit's shipment to the customer. Therefore, the access to *normal* fingerprint data is limited, driving the demand for an embedded application of GMM and one-class classification. An embedded version of the ONDS is successfully ran on MKS RF power generators.

For the application of novelty detection of equine behavior, metrics are reported using normal behavior models for individual horses as well as generic models. A subset of target behaviors representing novel events (i.e., falling, rolling, rising, and shaking) were expertly identified from a custom dataset consisting of sensor data from five unique horses across three different nights. Using these target behaviors and a generic behavior model, the ONDS correctly identified 97.5% of them as novel. Furthermore, only 334 novel events per unit were detected across the three days with an average event time of 43.5 s. Individual models were also considered and shown to provide higher classification performance, but with significantly more novel events. Thus, given the dataset currently available it is suggested to use generic behavior models over individual models for applications requiring an embedded solution or limited processing power. With respect to processing power, the novel event detection algorithm was implemented and tested within an embedded system and a non-optimized Octave version that confirms fast execution times ~ 0.2 s.

Future work with the ONDS may involve online adaption of the GMM models. An online adaptation of the GMM models could allow for model adjustments to subtle systemic changes that may be introduced via system aging. By adapting the model, the number of false positives may be reduced. Furthermore, investigation into using supervised feedback to help improve the specificity of the GMM may be explored. Supervised feedback may be provided by an expert to confirm or reject the one-class classifiers decision. Both applications of RF power generators and equine novel behavior detection would benefit from online adaptation and supervised feedback. In regards to ONDS and equine novel behavior detection, future work may involve the integration of the ONDS and other secondary processing methods. These secondary processing methods are intended to classify novel events for high-level behavioral detection and quantification of equine distress. One potential method for secondary processing is the use of multiple one-class GMM models to represent individual targeted behaviors. Using the existing dataset, each behavior would have its own model. Novel events could be compared against each model, and the model with highest likelihood selected for the behavior's classification. This use of individual GMMs would allow for software reuse to maintain a small footprint and feasibly of such an embedded system.

The results collected from the exploration of preprocessing methods and parameter selection have confirmed the specific methods that have been implemented for an embedded application. Once a GMM is learned, the online-version of the one-class classification algorithm within an embedded system is feasible.

Appendix A

Simplification and Derivations of MLE

A.1 Simplification - Univariate Gaussian Distribution

$$\mathcal{L}\left(\mu, \sigma \,|\, \mathbf{y}\right) = log \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y_j - \mu\right)^2}{2\sigma^2}\right) \tag{A.1}$$

$$\mathcal{L}\left(\mu, \sigma \,|\, \mathbf{y}\right) = log \left[\frac{\left(2\pi\right)^{-\frac{N}{2}}}{\sigma^N} \exp\left(-\sum_{j=1}^{N} \frac{\left(y_j - \mu\right)^2}{2\sigma^2}\right)\right] \tag{A.2}$$

$$\mathcal{L}\left(\mu, \sigma \,|\, \mathbf{y}\right) = log \left[\frac{\left(2\pi\right)^{-\frac{N}{2}}}{\sigma^N}\right] + \log\left[\exp\left(-\sum_{j=1}^{N} \frac{\left(y_j - \mu\right)^2}{2\sigma^2}\right)\right] \tag{A.3}$$

$$\mathcal{L}\left(\mu, \sigma \,|\, \mathbf{y}\right) = \log\left[\left(2\pi\right)^{-\frac{N}{2}}\right] - \log\left[\sigma^N\right] - \sum_{j=1}^{N} \frac{\left(y_j - \mu\right)^2}{2\sigma^2} \tag{A.4}$$

$$\mathcal{L}\left(\mu, \sigma \,|\, \mathbf{y}\right) = -\frac{N}{2}\log\left[2\pi\right] - N\log\left[\sigma\right] - \sum_{j=1}^{N} \frac{\left(y_j - \mu\right)^2}{2\sigma^2} \tag{A.5}$$

A.2 Simplification - Multivariate Gaussian Distribution

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \log \prod_{j=1}^{N} \frac{1}{\left(2\pi\right)^{M/2} \left|\boldsymbol{\Sigma}\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right] \tag{A.6}$$

Let $\Lambda = \boldsymbol{\Sigma}^{-1}$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \log \prod_{j=1}^{N} \left(2\pi\right)^{-\frac{M}{2}} \left|\boldsymbol{\Lambda}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right) \tag{A.7}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \sum_{j=1}^{N} \log\left[\left(2\pi\right)^{-\frac{M}{2}} \left|\boldsymbol{\Lambda}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right)\right] \tag{A.8}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = \sum_{j=1}^{N} \log\left[\left(2\pi\right)^{-\frac{M}{2}} \left|\boldsymbol{\Lambda}\right|^{\frac{1}{2}}\right] + \sum_{j=1}^{N} \log\left[\exp\left(-\frac{1}{2}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right)\right)\right] \tag{A.9}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = N\left(\log\left[\left(2\pi\right)^{-\frac{M}{2}} \left|\boldsymbol{\Lambda}\right|^{\frac{1}{2}}\right]\right) - \frac{1}{2}\sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{A.10}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = N\left(\log\left[\left(2\pi\right)^{-\frac{M}{2}}\right] + \log\left[\left|\boldsymbol{\Lambda}\right|^{\frac{1}{2}}\right]\right) - \frac{1}{2}\sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda}\left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{A.11}$$

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma} \,|\, \mathbf{Y}\right) = -\frac{N \cdot M}{2} \log\left(2\pi\right) + \frac{N}{2} \log\left|\boldsymbol{\Lambda}\right| - \frac{1}{2} \sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda} \left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{A.12}$$

A.3 Derivation - Multivariate Gaussian Distribution

Let $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ be the precision matrix (inverse of the covariance matrix) and further simplification yields the log likelihood of the MVN to be:

$$\mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = -\frac{N \cdot M}{2} \log\left(2\pi\right) + \frac{N}{2} \log\left|\boldsymbol{\Lambda}\right| - \frac{1}{2} \sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda} \left(\mathbf{y}_j - \boldsymbol{\mu}\right) \tag{A.13}$$

To find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}$ using MLE is to solve:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = 0 \text{ and } \frac{\partial}{\partial \boldsymbol{\Lambda}} \mathcal{L}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda} \,|\, \mathbf{Y}\right) = 0 \tag{A.14}$$

For $\hat{\boldsymbol{\mu}}$ step-wise solve:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \left[-\frac{N \cdot M}{2} \log\left(2\pi\right) + \frac{N}{2} \log\left|\boldsymbol{\Lambda}\right| - \frac{1}{2} \sum_{j=1}^{N} \left(\mathbf{y}_j - \boldsymbol{\mu}\right)^T \boldsymbol{\Lambda} \left(\mathbf{y}_j - \boldsymbol{\mu}\right) \right] = 0 \tag{A.15}$$

Substitution: $\mathbf{x}_j = \left(\mathbf{y}_j - \boldsymbol{\mu}\right) \implies \dfrac{\partial \mathbf{x}_j}{\partial \boldsymbol{\mu}} = -1$

$$-\frac{1}{2} \sum_{j=1}^{N} \frac{\partial}{\partial \mathbf{x}_j} \frac{\partial \mathbf{x}_j}{\partial \boldsymbol{\mu}} \mathbf{x}_j^T \boldsymbol{\Lambda} \mathbf{x}_j = 0 \tag{A.16}$$

Recall: $\frac{\partial\left(a^T A a\right)}{\partial a} = \left(A + A^T\right) a$

$$-\frac{1}{2}\sum_{j=1}^{N} -1\left(\mathbf{\Lambda} + \mathbf{\Lambda}^T\right)\mathbf{x}_j = 0 \tag{A.17}$$

Re-substitute: $\mathbf{\Sigma}^{-1} = \mathbf{\Lambda}$ and $(\mathbf{y}_j - \boldsymbol{\mu}) = \mathbf{x}_j$

$$-\frac{1}{2}\sum_{j=1}^{N} -1\left(\mathbf{\Sigma}^{-1} + \mathbf{\Sigma}^{-T}\right)(\mathbf{y}_j - \boldsymbol{\mu}) = 0 \tag{A.18}$$

Recall that $\mathbf{\Sigma}$ is symmetric and $A^{-T} = A^{-1}$ for a symmetric matrix:

$$-\frac{1}{2}\sum_{j=1}^{N} -2\mathbf{\Sigma}^{-1}(\mathbf{y}_j - \boldsymbol{\mu}) = \mathbf{\Sigma}^{-1}\sum_{j=1}^{N}(\mathbf{y}_j - \boldsymbol{\mu}) = 0 \tag{A.19}$$

Further simplification and re-arrangement will yield the estimator $\hat{\boldsymbol{\mu}}$ as:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{j=1}^{N}\mathbf{y}_j \tag{A.20}$$

For $\hat{\mathbf{\Sigma}}$ step-wise solve still using $\mathbf{\Lambda} = \mathbf{\Sigma}^{-1}$:

$$\frac{\partial}{\partial \mathbf{\Lambda}}\left[-\frac{N \cdot M}{2}\log(2\pi) + \frac{N}{2}\log|\mathbf{\Lambda}| - \frac{1}{2}\sum_{j=1}^{N}(\mathbf{y}_j - \boldsymbol{\mu})^T \mathbf{\Lambda}(\mathbf{y}_j - \boldsymbol{\mu})\right] = 0 \tag{A.21}$$

Use the trace "trick": $a^T A a = \operatorname{tr}\left(a^T A a\right) = \operatorname{tr}\left(a a^T A\right)$

$$\frac{\partial}{\partial \mathbf{\Lambda}}\left[-\frac{N \cdot M}{2}\log(2\pi) + \frac{N}{2}\log|\mathbf{\Lambda}| - \frac{1}{2}\sum_{j=1}^{N}\operatorname{tr}\left((\mathbf{y}_j - \boldsymbol{\mu})(\mathbf{y}_j - \boldsymbol{\mu})^T \mathbf{\Lambda}\right)\right] = 0 \tag{A.22}$$

Substitute: $\mathbf{S} = \sum_{j=1}^{N} (\mathbf{y}_j - \hat{\boldsymbol{\mu}}) (\mathbf{y}_j - \hat{\boldsymbol{\mu}})^T$

$$\frac{\partial}{\partial \boldsymbol{\Lambda}} \left[-\frac{N \cdots M}{2} \log(2\pi) + \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \mathrm{tr}(\mathbf{S}\boldsymbol{\Lambda}) \right] = 0 \tag{A.23}$$

Distribute the partial derivative:

$$\frac{N}{2} \frac{\partial}{\partial \boldsymbol{\Lambda}} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \frac{\partial}{\partial \boldsymbol{\Lambda}} \mathrm{tr}(\mathbf{S}\boldsymbol{\Lambda}) = 0 \tag{A.24}$$

Recall $\frac{\partial}{\partial A} \mathrm{tr}(BA) = B^T$ and $\frac{\partial}{\partial A} \log |A| = A^{-T}$:

$$\frac{N}{2} \boldsymbol{\Lambda}^{-T} - \frac{1}{2} \mathbf{S}^T = 0 \tag{A.25}$$

Rearrange:

$$\boldsymbol{\Lambda}^{-1} = \frac{1}{N} \mathbf{S} \tag{A.26}$$

Re-substitute $\boldsymbol{\Sigma}^{-1} = \boldsymbol{\Lambda}$ and $\sum_{j=1}^{N} (\mathbf{y}_j - \hat{\boldsymbol{\mu}}) (\mathbf{y}_j - \hat{\boldsymbol{\mu}})^T = \mathbf{S}$ and the estimator $\hat{\boldsymbol{\Sigma}}$ is:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{j=1}^{N} (\mathbf{y}_j - \hat{\boldsymbol{\mu}}) (\mathbf{y}_j - \hat{\boldsymbol{\mu}})^T \tag{A.27}$$

References

[1] R.M. Bowen, Ferat Sahin, and Jeffrey Schab. Early identification of equine distress: Novel event detection of behavior. *Journal of Equine Veterinary Science*, submitted for publication.

[2] R.M. Bowen, F. Sahin, and Radomski Aaron. Systemic health evaluation on rf generators using gaussian mixture models. *Computers and Electrical Engineering*, accepted for publication.

[3] Ryan M Bowen, Ferat Sahin, Aaron Radomski, and Dan Sarosky. Embedded one-class classification on rf generator using mixture of gaussians. In *2014 IEEE International Conference on Systems, Man, and Cybernetics*, page TBD, October 2014.

[4] Anne Raich and Ali Cinar. Statistical process monitoring and disturbance diagnosis in multivariable continuous processes. *AIChE Journal*, 42(4):995–1009, 1996.

[5] Brian E Goodlin, Duane S Boning, Herbert H Sawin, and Barry M Wise. Simultaneous fault detection and classification for semiconductor manufacturing tools. *Journal of the Electrochemical Society*, 150(12):G778–G784, 2003.

[6] Sang Jeen Hong, Woo Yup Lim, Taesu Cheong, and Gary S May. Fault detection and classification in plasma etch equipment for semiconductor manufacturing-diagnostics. *Semiconductor Manufacturing, IEEE Transactions on*, 25(1):83–93, 2012.

[7] Anna Maria Ison. *Probabilistic Modeling for Fault Classification of Plasma Equipment*. PhD thesis, University of California, Berkeley, 1999.

[8] Yuan Li and Xinmin Zhang. Diffusion maps based k-nearest-neighbor rule technique for semiconductor manufacturing process fault detection. *Chemometrics and Intelligent Laboratory Systems*, 136(0):47 – 57, 2014.

[9] Jianbo Yu. Semiconductor manufacturing process monitoring using gaussian mixture model and bayesian method with local and nonlocal information. *Semiconductor Manufacturing, IEEE Transactions on*, 25(3):480–493, Aug 2012.

[10] Sankar Mahadevan and Sirish L. Shah. Fault detection and diagnosis in process data using one-class support vector machines. *Journal of Process Control*, 19(10):1627 – 1639, 2009.

[11] Jonghyuck Park, Ick-Hyun Kwon, Sung-Shick Kim, and Jun-Geol Baek. Spline regression based feature extraction for semiconductor process fault detection using support vector machine. *Expert Systems with Applications*, 38(5):5711 – 5718, 2011.

[12] Anna M Ison, Wei Li, and Costas J Spanos. Fault diagnosis of plasma etch equipment. In *Semiconductor Manufacturing Conference Proceedings, 1997 IEEE International Symposium on*, pages B49–B52. IEEE, 1997.

[13] Chen-Fu Chien, Chia-Yu Hsu, and Pei-Nong Chen. Semiconductor fault detection and classification for yield enhancement and manufacturing intelligence. *Flexible Services and Manufacturing Journal*, 25(3):367–388, 2013.

[14] A Ordaz-Moreno, R. de Jesus Romero-Troncoso, J.A Vite-Frias, J.R. Rivera-Gillen, and A Garcia-Perez. Automatic online diagnosis algorithm for broken-bar detection on induction motors based on discrete wavelet transform for fpga implementation. *Industrial Electronics, IEEE Transactions on*, 55(5):2193–2202, May 2008.

[15] Michael W Vanik, JL Beck, and SK2000 Au. Bayesian probabilistic approach to structural health monitoring. *Journal of Engineering Mechanics*, 126(7):738–745, 2000.

[16] Yu Wang, Qiang Miao, E.W.M. Ma, Kwok-Leung Tsui, and M.G. Pecht. Online anomaly detection for hard disk drives based on mahalanobis distance. *Reliability, IEEE Transactions on*, 62(1):136–145, March 2013.

[17] Piyush Shakya, Makarand S. Kulkarni, and Ashish K. Darpe. A novel methodology for online detection of bearing health status for naturally progressing defect. *Journal of Sound and Vibration*, 333(21):5614 – 5629, 2014.

[18] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.

[19] J. Kivinen, AJ. Smola, and R.C. Williamson. Online learning with kernels. *Signal Processing, IEEE Transactions on*, 52(8):2165–2176, Aug 2004.

[20] Wenjian Wang, Changqian Men, and Weizhen Lu. Online prediction model based on support vector machine. *Neurocomputing*, 71(4):550–558, 2008.

[21] Okba Taouali, Ilyes Elaissi, and Hassani Messaoud. Online prediction model based on reduced kernel principal component analysis. In *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on*, pages 1–6. IEEE, 2009.

[22] Ilyes Elaissi, Ines Jaffel, Okba Taouali, and Hassani Messaoud. Online prediction model based on the svdkpca method. {*ISA*} *Transactions*, 52(1):96 – 104, 2013.

[23] L.G. Sun, C.C. de Visser, Q.P. Chu, and J.A. Mulder. A novel online adaptive kernel method with kernel centers determined by a support vector regression approach. *Neurocomputing*, 124(0):111 – 119, 2014.

[24] LD Moya, MM and Koch, MW and Hostetler. One-class classifier networks for target recognition applications. *NASA STI/Recon Technical Report N*, 93:24043, 1993.

[25] D.M.J Tax. *One-Class Classification: Concept-learning in the Absence of Counter-examples*. PhD thesis, Delft University of Technology, 2001.

[26] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *ACM Computing Surveys*, 41(3):1–58, July 2009.

[27] Markos Markou and Sameer Singh. Novelty detection: a review part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, December 2003.

[28] Nathalie Japkowicz. *Concept -learngin in the absence of counter-examples: An autoassociation-based approach to classification*. PhD thesis, Rutgers, 1999.

[29] Markos Markou and Sameer Singh. Novelty detection: a review part 2: neural network based approaches. *Signal Processing*, 83(12):2499–2521, December 2003.

[30] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, June 2014.

[31] DavidM.J. Tax and RobertP.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[32] Xuemei Ding, Yuhua Li, Ammar Belatreche, and Liam P. Maguire. An experimental evaluation of novelty detection methods. *Neurocomputing*, 135(0):313 – 327, 2014. Advances in Learning Schemes for Function Approximation Selected papers from the 11th International Conference on Intelligent Systems Design and Applications (ISDA 2011).

[33] D.P. Filev, R.B. Chinnam, F. Tseng, and P. Baruah. An industrial strength novelty detection framework for autonomous equipment monitoring and diagnostics. *Industrial Informatics, IEEE Transactions on*, 6(4):767–779, Nov 2010.

[34] Dimitar Filev and Plamen Angelov. *Algorithms for Real-time Clustering and Generation of Rules from Data*, pages 354–369. John Wiley & Sons, Ltd, 2007.

[35] P.P. Angelov and D.P. Filev. An approach to online identification of takagi-sugeno fuzzy models. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(1):484–498, Feb 2004.

[36] J J Verbeek, N Vlassis, and B Kröse. Efficient greedy learning of gaussian mixture models. *Neural computation*, 15(2):469–85, February 2003.

[37] Geoffrey McLachlan and David Peel. *Finite mixture models*. John Wiley & Sons, 2004.

[38] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[39] Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In Emanuel Parzen, Kunio Tanabe, and Genshiro Kitagawa, editors, *Selected Papers of Hirotugu Akaike*, Springer Series in Statistics, pages 199–213. Springer New York, 1998.

[40] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[41] David A Dickey and Wayne A Fuller. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica: Journal of the Econometric Society*, pages 1057–1072, 1981.

[42] Nikos Vlassis and Aristidis Likas. A greedy em algorithm for gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87, 2002.

[43] Franz Pernkopf and Djamel Bouchaffra. Genetic-based em algorithm for learning gaussian mixture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1344–1348, 2005.

[44] Wolfgang Jank. Ascent em for fast and global solutions to finite mixtures: An application to curve-clustering of online auctions. *Computational statistics & data analysis*, 51(2):747–761, 2006.

[45] Koji Kashihara. Automatic design of a novel image filter based on the ga-em algorithm for vein shapes. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 3897–3902. IEEE, 2013.

[46] Aleix M Martınez and Jordi Vitria. Learning mixture models using a genetic version of the em algorithm. *Pattern Recognition Letters*, 21(8):759–769, 2000.

[47] Jussi Tohka, Evgeny Krestyannikov, Ivo D Dinov, AM Graham, David W Shattuck, Ulla Ruotsalainen, and Arthur W Toga. Genetic algorithms for finite mixture model based voxel classification in neuroimaging. *Medical Imaging, IEEE Transactions on*, 26(5):696–711, 2007.

[48] Jing-Hua Guan, Da-You Liu, and Si-Pei Liu. Discrete particle swarm optimization and em hybrid approach for naive bayes clustering. In Irwin King, Jun Wang, Lai-Wan Chan, and DeLiang Wang, editors, *Neural Information Processing*, volume 4233 of *Lecture Notes in Computer Science*, pages 1164–1173. Springer Berlin Heidelberg, 2006.

[49] Bing Zhang, Lina Zhuang, Lianru Gao, Wenfei Luo, Qiong Ran, and Qian Du. Pso-em: A hyperspectral unmixing algorithm based on normal compositional model. *Geoscience and Remote Sensing, IEEE Transactions on*, 52(12):7782–7792, Dec 2014.

[50] Yanjun Yan and LisaAnn Osadciw. Density estimation using a new dimension adaptive particle swarm optimization algorithm. *Swarm Intelligence*, 3(4):275–301, 2009.

[51] Alan V Oppenheim, Ronald W Schafer, John R Buck, et al. *Discrete-time signal processing*, volume 2. Prentice-hall Englewood Cliffs, 1989.

[52] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[53] Peter D Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.

[54] Ethem Alpaydn. *Introduction to Machine Learning.* The MIT Press, Cambridge, Massachusetts, second edition, 2010.

[55] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[56] Ethem Alpaydin. *Introduction to Machine Learning.* MIT Press, Cambirdge, Massachusetts, 2004.

[57] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[58] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society . Series B (Methodological)*, 39(1):1–38, 1977.

[59] Geoffrey J. Mclachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions.* Wiley-Interscience, 1996.

[60] Charles E. Metz. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, 8(4):283 – 298, 1978.

[61] Aaron T. Radomski and Kevin P. Nasman. Controller for rf power generator with reduced cable length sensitivity, July 2002.

[62] Michael A. Lieberman and Allan J. Lichtenburg. *Principles of Plasma Discharges and Materials Processing.* John Wiley & Sons, Inc., Hoboken, New Jersey, second edition, 2005.

[63] Girish Chandrashekar, Ferat Sahin, Eyup Cinar, Aaron Radomski, and Dan Sarosky. In-vivo fault analysis and real-time fault prediction for rf generators using state-of-the-art classifiers. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK, 2013.

[64] Girish Chandrashekar and Ferat Sahin. In-vivo fault prediction for rf generators using variable elimination and state-of-the-art classifiers. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1800–1805, Seoul, Korea, October 2012. Ieee.

[65] M. Lichman. UCI Machine Learning Repository, url = http://archive.ics.uci.edu/m, 2013.

[66] W H Wolberg and O L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, 87(23):9193–9196, 1990.

[67] Asli Celikyilmaz and I. Burhan Turksen. Fuzzy functions with support vector machines. *Information Sciences*, 177(23):5163 – 5177, 2007.

[68] Asli Celikyilmaz, I. Burhan Turksen, Ramazan Aktas, M. Mete Doganay, and N. Basak Ceylan. Increasing accuracy of two-class pattern recognition with enhanced fuzzy functions. *Expert Systems with Applications*, 36(2, Part 1):1337 – 1354, 2009.

[69] Eyup Cinar and Ferat Sahin. New classification techniques for electroencephalogram (eeg) signals and a real-time eeg control of a robot. *Neural Computing and Applications*, 22(1):29–39, 2013.

[70] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[71] Help lower your horses risk of colic during show season. *NRHA Reiner*, pages 138–139, May 2014.

[72] Colic: An age-old problem. *CEH Horse Report*, 26(1):1–7, 2008.

[73] Vanessa L. Cook and Diana M. Hassel. Evaluation of the colic in horses: Decision for referral. *Veterinary Clinics of North America: Equine Practice*, 30(2):383 – 398, 2014. Emergency and Critical Care.

[74] Colic: Minimizing its incidence and impact in your horse. *AAEP Horse Health*, 2010.

[75] C. E. Scantlebury, D. C. Archer, C. J. Proudman, and G. L. Pinchbeck. Recurrent colic in the horse: Incidence and risk factors for recurrence in the general practice population. *Equine Veterinary Journal*, 43:81–88, 2011.

[76] C. E. Scantlebury, D. C. Archer, C. J. Proudman, and G. L. Pinchbeck. Management and horse-level risk factors for recurrent colic in the uk general equine practice population. *Equine Veterinary Journal*, 47(2):202–206, 2015.

[77] Roland E. Thomas, Albert J. Rosa, and Gregory J. Toussaint. *The Analysis and Design of Linear Circuits*. Wiley Publishing, 7th edition, 2011.

[78] MATLAB. *version 8.5.0 (R2015a)*. The MathWorks Inc., Natick, Massachusetts, 2015.

[79] John W. Eaton, David Bateman, and Soren Hauberg. *GNU Octave version 3.0.1 manual: a high-level interactive language for numerical computations*. CreateSpace Independent Publishing Platform, 2009. ISBN 1441413006.