Anomaly Detection Methods for Unmanned Underwater Vehicle Performance Data

by

William Ray Harris

B.S., Mathematics US Naval Academy (2013)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author ..... Signature redacted

Sloan School of Management
May 15, 2014

Certified by ..... Signature redacted

Dr. Michael J. Ricard
Laboratory Technical Staff
Charles Stark Draper Laboratory, Inc.
Thesis Supervisor

Certified by ..... Signature redacted

Prof. Cynthia Rudin
Associate Professor of Statistics
MIT CSAIL and Sloan School of Management
Thesis Supervisor

Accepted by ..... Signature redacted .....

Prof. Dimitris Bertsimas
Boeing Professor of Operations Research
Co-Director, Operations Research Center

THIS PAGE INTENTIONALLY LEFT BLANK

Anomaly Detection Methods for Unmanned Underwater Vehicle Performance Data

by

William Ray Harris

Submitted to the Sloan School of Management
on May 15, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

This thesis considers the problem of detecting anomalies in performance data for unmanned underwater vehicles(UUVs). UUVs collect a tremendous amount of data, which operators are required to analyze between missions to determine if vehicle systems are functioning properly. Operators are typically under heavy time constraints when performing this data analysis. The goal of this research is to provide operators with a post-mission data analysis tool that automatically identifies anomalous features of performance data. Such anomalies are of interest because they are often the result of an abnormal condition that may prevent the vehicle from performing its programmed mission. In this thesis, we consider existing one-class classification anomaly detection techniques since labeled training data from the anomalous class is not readily available. Specifically, we focus on two anomaly detection techniques: (1) Kernel Density Estimation (KDE) Anomaly Detection and (2) Local Outlier Factor. Results are presented for selected UUV systems and data features, and initial findings provide insight into the effectiveness of these algorithms. Lastly, we explore ways to extend our KDE anomaly detection algorithm for various tasks, such as finding anomalies in discrete data and identifying anomalous trends in time-series data.

Thesis Supervisor: Dr. Michael J. Ricard
Title: Laboratory Technical Staff
Charles Stark Draper Laboratory, Inc.

Thesis Supervisor: Prof. Cynthia Rudin
Title: Associate Professor of Statistics
MIT CSAIL and Sloan School of Management

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

There are many to whom I owe thanks for both the completion of this thesis and my experience at MIT. Firstly, I would like to thank Dr. Michael Ricard of Draper Laboratory. His support over the past two years has been invaluable. His genuine concern for my well-being and personal success has made my time at MIT a very rewarding experience.

I would like to thank the Navy, Draper Laboratory, and the Operations Research Center for giving me the opportunity to pursue a Master's degree. Studying operations research at MIT has been an amazing educational opportunity that I believe will forever enrich my life and career.

I am also grateful to Dr. Cynthia Rudin for her guidance and support. She has been a tremendous resource for all of the roadblocks that I encountered throughout the completion of this thesis.

I would also like to thank employees of NUWC and WHOI for taking the time to help me with my research. In particular, I would like to thank Tom Merchant and Nathan Banks, along with others at NUWC for sharing their technical knowledge of the UUVs that I researched for this thesis. I would also like to thank Tom Austin, Mike Purcell, and Roger Stokey of WHOI for providing the data and software used in this research, and also for sharing their technical knowledge of UUVs.

Lastly, I would not have had such a great experience at MIT without the love and support of my family and friends. Classes at MIT were an enjoyable experience with the help of my friends in the ORC. I would also like to thank Rob, Casey, and Amy, my wonderful office mates at Draper Lab, for their friendship and support.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of Charles Stark Draper Laboratory, the United States Navy, Department of Defense, or the U.S. Government.

William Ray Harris, ENS, USN 15 May 2015

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Motivation

Unmanned underwater vehicles (UUVs) are utilized both commercially and by the military for a variety of mission categories, including oceanography, mine countermeasures (MCM), and intelligence, surveillance and reconnaissance (ISR) [6]. Many of these mission categories require that UUVs traverse as much area as possible, and, due to resource constraints, in a limited amount of time. As a result, UUV operators will typically seek to minimize the turnaround time between missions in order to maximize the amount of time that a vehicle is in the water and performing its programmed mission.

One constraint when attempting to minimize the turnaround time between sorties is the need for operators to perform post-mission data analysis. UUVs collect large amounts of data on every mission. The collected data often contains hundreds of variables from a variety of different sensors, and each variable typically contains thousands of time-series data points. Due to the limitations of underwater communications, operators are often unable to retrieve this data in real time. Thus, data analysis must be performed post-mission, after the vehicle has come to the surface and data has been uploaded.

Post-mission data analysis is performed to ensure that the vehicle is behaving as expected. Operators must check that the vehicle has adequately performed its

programmed mission, and that the vehicle is in good working condition. The task requires an experienced operator who can distinguish between "normal" and "faulty" data. Due to time constraints and the volume of data, this can be a difficult task, even for experienced UUV operators. In addition, it can be difficult to identify anomalous long-term trends in performance data. For example, a certain system may experience degradation over the period of 50 missions. This degradation may be difficult to identify by operators analyzing data from individual missions.

We would like to be able to automate much of the post-mission data analysis task and provide operators with a tool to help identify anomalous features of performance data, since anomalous data may be indicative of problem with the vehicle. The desired benefits of such a tool are as follows:

- Improved ability of detecting faults in UUV systems.
- Decreased turnaround time between sorties.
- Less operator experience required for post-mission data analysis.

In short, a reliable post-mission analysis tool means less resources (e.g. time, money) being spent per UUV mission, and improved reliability of UUV performance.

1.2 General Problem Statement

This thesis considers the problem of detecting anomalies in UUV performance data. *Anomalies* are defined as "patterns in data that do not conform to a well defined notion of normal behavior"[9]. The assumption in this case is that anomalous data is often indicative of a *fault*. We define fault, as in [22], to be "an abnormal condition that can cause an element or an item to fail". By identifying anomalous data, we can alert UUV operators towards potential faults.

Specifically, we focus on identifying anomalous features of continuous time-series data. Our goal is to build a classifier, trained on historical data, that can distinguish between "normal" and "anomalous" data instances. Our goal is to classify a particular subsystem for a given mission as either anomalous or normal. For this thesis, the

term "normal" will always refer to expected or regular behavior with respect to data instances. We will use the term Gaussian to referring to the probability distribution commonly known as the normal distribution.

1.3 Approach

The problem of anomaly detection is heavily dependent on both the nature of available data and the types of anomalies to be detected [9]. In this thesis, we explore existing anomaly detection techniques, and we discuss which methods can be effectively applied to UUV performance data.

We will take a one-class classification approach to identifying anomalous data features for a particular UUV subsystem for a given mission. One-class classification anomaly detection techniques assume that all training data is from the same class (e.g. the normal class)[9]. With this approach, we attempt to build a classifier based on normal training data. One issue in this case is that we do not have readily available training data. That is, we do not know whether or not UUV performance data for a given mission contains unmodeled faults. The key assumption that we make is that the majority of UUV missions that we use for training models do not contain faults for the systems of interest.

As will be discussed in Chapter 4, we determine that anomaly detection using kernel density estimation (KDE) is well suited for the problem of UUV anomaly detection. Hence, we will primarily focus on the KDE anomaly detection algorithm for our experimentation. The basic approach of this technique is to develop a probability model for the data generating process of features of UUV performance data. We classify future data instances as anomalous if they lie in areas of low probability of our stochastic model.

1.4 Contributions

The contributions of this thesis are as follows:

1. A survey of one class classification methods that are applicable to the problem of UUV anomaly detection.

2. A discussion of parameter selection for our KDE anomaly detection algorithm.

3. Development of an algorithm for computing decision boundaries based on our KDE models.

4. Experimental results for selected data features for UUV subsystems. We compare the KDE anomaly detection algorithm with another one-class classification method called Local Outlier Factor (LOF).

5. A discussion of extending our KDE algorithm for various purposes, namely, (1) detecting anomalies in discrete data, (2) incorporating new data into our KDE models while accounting for potential anomalous trends, and (3) using previous KDE models for new UUVs.

1.5 Thesis Organization

Chapter 2 of this thesis gives an overview of the UUVs and data sets that were used in this research. Chapter 3 provides a description of the anomaly detection problem in its most general form. In this chapter, we discuss the characteristics of the UUV anomaly detection problem, and conclude that one-class classification methods are suitable for this problem. In Chapter 4, we explore existing one-class classification methods that may be suitable for UUV anomaly detection. In chapter 5, we discuss parameter selection for our KDE anomaly detection algorithm, and we also demonstrate how to compute decision boundaries for our classifier. In Chapter 6, we show experimental results for selected UUV subsystems and data features. Lastly, Chapter 7 is a discussion of extending our KDE anomaly detection algorithm for various purposes.

Chapter 2

UUV Background

The UUVs discussed in this thesis are the *Remote Environmental Monitoring Unit* (REMUS) series of vehicles and the *Mid-Sized Autonomous Reconfigurable Vehicle* (MARV). In this section, we will discuss the systems and sensor payloads of these vehicles, as well as operational capabilities. We will also describe the data sets that were available for this research.

2.1 REMUS

The REMUS series of vehicles was developed by Woods Hole Oceanographic Institute (WHOI), and is currently manufactured by Hydroid, Inc. The goal of the REMUS project was to develop a UUV that could collect accurate oceanographic measurements over a wide area, at a small cost. The first REMUS was built in 1995[15].

Today, the REMUS series consists of the six types vehicles of varying sizes, specifications, and capabilities. In this research, we had access to data from two types of vehicles, namely the REMUS 600 and the REMUS 6000. The specifications of these two vehicles are given in table 2.1. The most notable difference is that the REMUS 6000 is larger, and capable of operating at depths of up to 6000 meters [15].

REMUS vehicles maneuver using a propeller and fins. They are powered by rechargeable lithium ion batteries, which must either be recharged or replaced after a mission. There are several ways that a REMUS can obtain a position fix, which

Parameter	**REMUS 600**	**REMUS 6000**
Diameter	12.75 inches	28 inches
Weight	530 lbs	1900 lbs
Max Depth	600 meters (1968.5 feet)	6000 meters
Max Endurance	70 hours	22 hours

Table 2.1: REMUS 600 and REMUS 6000 Vehicle Specifications

is an estimate of the vehicle's position in the water. The most accurate method is through GPS, but this requires that the vehicle is on the surface since GPS signals will not travel through water. When the vehicle is beneath the surface, it can recieve an acoustic fix by measuring distances from underwater transponders. The position of these transponders is known, and the vehicle can measure distance to and from the transponders by measuring the time that an acoustic ping takes to reach the transponder. Lastly, the vehicle can use dead reckoning using sensor measurements from the Inertial Navigation System (INS) and Acoustic Doppler Current Profiling (ADCP). Dead reckoning is the process of estimating the vehicles position by advancing position from a previous fix based on measurements of velocity and angle through the water.

In addition to the INS and ADCP, the REMUS can be fitted with a variety of sensors, depending on mission requirements. Sensors that are commonly utilized include side scan sonar, conductivity & temperature, acoustic imaging, and sub-bottom profiling sensors. The ability to customize the REMUS sensor payload allows the vehicle to perform a variety of missions. These missions can be programmed on a laptop computer using the REMUS Vehicle Interface Program (VIP). Typical REMUS applications include[16]:

- Mine Counter Measure Operations
- Hydrographic Surveying
- Search & Salvage Operations
- Scientific Sampling & Mapping
- Debris Field Mapping

Figure 2-1: REMUS 600 - Autonomous Underwater Vehicle[1]

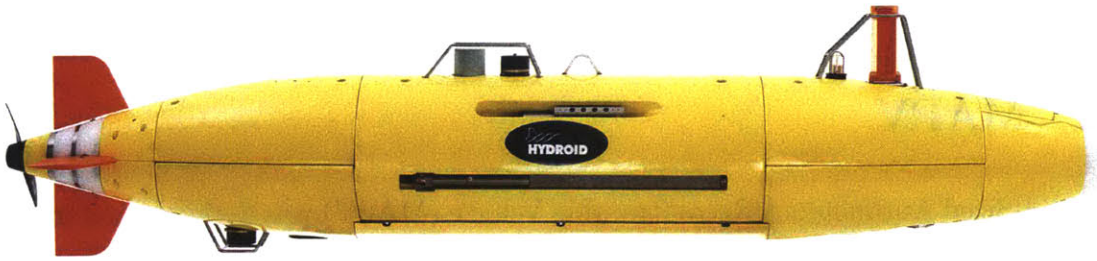


Figure 2-2: REMUS 6000 - Autonomous Underwater Vehicle[1]

REMUS Data Sets

There are two collections of REMUS mission data that were obtained for this research. The first collection is from three separate REMUS 600 vehicles that combined to perform 282 missions between 2011 and 2014. This mission data was obtained from the Naval Undersea Warfare Center, Division Newport (NUWC-NPT). The second collection of mission data is from two REMUS 6000 vehicles. The data covers 72 deep water surveying missions conducted by WHOI operators in 2009.

The data available for each mission is similar for both the REMUS 600 and REMUS 6000 collections. Each mission contains over 100 variables, and each variable contains thousands of data points that are in time-series. Of these variables, we are interested in those that a REMUS operator might analyze between missions to identify vehicle faults. According to REMUS operators at WHOI and NUWC-NPT, some variables that are typically analyzed after each mission are **thruster output**, **vehicle control data** (e.g. pitch, yaw, etc.), and **navigation performance data** (e.g. frequency of acoustic fixes). Hence, we will focus on identifying anomalous missions based on these variables.

In addition to the time-series data collected on each mission, the REMUS vehicle

maintains a **fault log**. This log contains a record of events that the REMUS is programmed to identify during a mission. Some of these are normal occurrences, such as the vehicle dropping an ascent weight before driving to the surface. Other events are modeled faults, such as the vehicle not be able to dive below the surface of the water. Again, for this thesis we are interested in identifying previously unmodeled faults, and not those that are identified in the fault log.

2.2 MARV

The Mid-Sized Autonomous Reconfigurable Vehicle (MARV) was designed and manufactured by the Naval Undersea Warfare Center. The MARV first became operational in 2004. The vehicle has similar systems, capabilities, and sensor payloads as the REMUS. One notable advantage of the MARV vehicle is that it has thruster control capabilities that allow the vehicle to hover. This allows the vehicle to remain in one place, which is ideal for object inspection or stand-off reconnaissance. The vehicle specifications for the MARV are given in table 2.2[7]. The vehicle is powered by a rechargeable lithium ion battery.

Parameter	**MARV**
Diameter	12.75 inches
Speed	2-5 knots
Max Depth	1500 feet
Endurance	10-24 hours

Table 2.2: MARV Vehicle Specifications

The MARV, like the REMUS, has a torpedo shape design (shown in figure 2-3). Also, like the REMUS, the MARV can be fitted with a variety of different sensors. Common sensor payloads include side-scan sonar, chemical sensors, bathymetric sonar, and video cameras. Typical missions performed by the MARV include the following[7]:

- Water Column Chemical Detection and Mapping
- Sonar Survey of Water Columns and Ocean Floor

- Object Inspection
- Stand-off Reconnaissance

MARV Data Set

The MARV data used in this research is from a collection of missions run by operators at NUWC-NPT. The collection contains 128 missions from 2013. Like the REMUS data, each mission contains over 100 variables with thousands of data points in time-series. Each mission contains sensor data as well as vehicle health data (e.g. battery status, thruster output, etc.).

Again, we will focus on variables that are typically analyzed for faults by operators between missions. These include variables related to thruster performance, navigation performance, and vehicle control.

Each MARV mission also generates a vehicle log and fault log. The vehicle log contains a record of each action that the vehicle takes on a mission. For example, it lists every change of direction and every acoustic fix that is obtained. The fault log, unlike the REMUS fault log, only lists a select few casualties, and is usually empty unless there is a fault that results in an aborted mission. Due to the fact that the vehicle log contains a much larger amount of information than the REMUS fault log, it is more difficult to pinpoint events that may be the cause of abnormal performance data for the MARV.

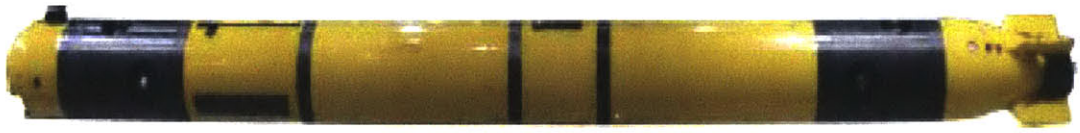


Figure 2-3: MARV - Autonomous Underwater Vehicle[2]

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Anomaly Detection Overview

This chapter provides an overview of the anomaly detection problem in its most general form. We discuss characteristics of anomaly detection problems and determine which existing techniques may be relevant to the problem of anomaly detection for UUV performance data.

3.1 Background

Anomaly detection, or *outlier detection*, is an important and well-researched problem with many applications. In existing literature, the terms anomaly and outlier are often used interchangeably. Hodge and Austin [13] define anomaly as "an observation that appears to deviate markedly from the other members of the sample in which it occurs". For this thesis, we use the definition given by Chandola, et al., in [9], "Anomalies are patterns in data that do not conform to a well defined notion of normal behavior". Approaches to anomaly detection are similar, and often identical, to those of novelty detection and noise detection.

Detecting anomalies is an important problem in many fields, since anomalous data instances often represent events of significance. For UUVs, anomalous data is often an indication of a mechanical fault or some other issue with the vehicle that would require action from an operator. Another example would be an anomalous spending patterns on a credit card. If a credit card is being used more frequently or in areas

outside of typical spending areas, it may be an indication of credit card fraud. Other examples of anomaly detection applications include the following[13]:

- **Intrusion Detection** - Identifying potential security breaches on computer networks.
- **Activity Monitoring** - Detecting fraud by monitoring relevant data (e.g. cell phone data or trade data).
- **Medical Condition Monitoring** - Detecting health problems as they occur (e.g. using heart-rate monitor data to detect heart problems).
- **Image Analysis** - Detecting features of interest in images (e.g. military targets in satellite images).
- **Text Data Analysis** - Identifying novel topics or events from news articles or other text data.

The unique attributes of each of these applications make the anomaly detection problem difficult to solve in general. Most anomaly detection techniques are catered toward a specific formulation of the problem. This formulation is determined by certain characteristics of our data and the types of anomalies we are interested in detecting. These characteristics are discussed in the following section.

3.2 Characteristics of Anomaly Detection Problems

Anomaly detection problems have been researched since as early as 1887 [10]. Since then, a large number of techniques have been introduced, researched, and tested. The necessity for a broad spectrum of techniques is due to unique characteristics of each application of anomaly detection.

In [9], Chandola, et al., list four aspects of anomaly detection problems that can affect the applicability of existing techniques. These four characteristics are **types of anomalies** that are of interest, the **nature of input data**, the availability of **data labels**, and the **output of detection**. In this section, we will address each of

these characteristics separately and discuss the characteristics of anomaly detection for UUV performance data.

3.2.1 Types of Anomalies

The first characteristic of an anomaly detection problem that we will address is the type of anomalies that are of interest. There are three categories of anomalies that we will discuss, namely point anomalies, contextual anomalies, and collective anomalies[9].

Point Anomalies

A point anomaly is a single data instance that does not fit to the remainder of the data. This is the simplest type of anomaly, and typically the easiest to detect.

Figure 3-1 shows a two-dimensional data set containing two point anomalies. The normal data is generated from a bivariate Gaussian distribution, and the point anomalies were inserted into areas of very low probability.

As an example for UUV data, consider sensor readings for a certain system, say thruster output. If the sensor readings give a single measurement that is outside of normal operational limits, then the measurement would be considered anomalous. Such an anomaly may be due to a problem with thrusters, or due to a sensor that is need of calibration. This is known as limit checking, and is the simplest method of anomaly detection for univariate data.

Contextual Anomalies

A contextual anomaly is a data instance that, given one or more contextual attributes, does not fit to the remainder of the data. Some examples of contextual attributes are time in time series data, position in sequential data, and latitude and longitude in spatial data. Having to account for contextual attributes makes the problem of identifying contextual anomalies more complex than that of point anomalies.

A simple example of a contextual anomaly would be an inch of snowfall in Texas

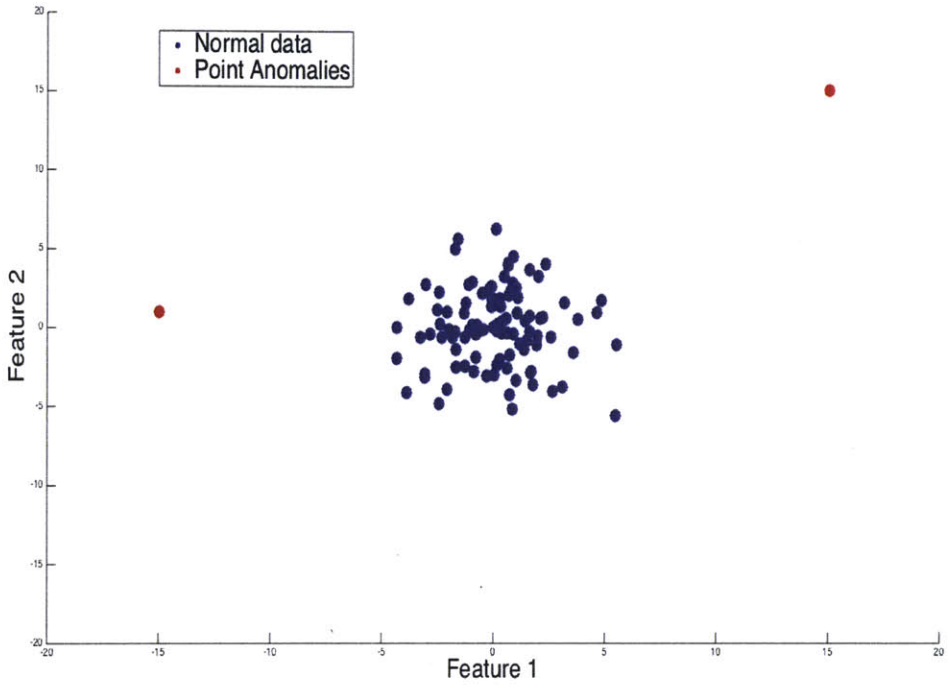


Figure 3-1: Illustration of point anomalies in artificially generated bivariate data

in April. An inch of snowfall is certainly not anomalous in many parts of the world during the month of April. Additionally, an inch of snowfall in Texas may not be considered anomalous in January. It is the combination of these contextual attributes (location and month) that make such a data instance anomalous. For UUV data, an example of a contextual anomaly would be a 10 degree roll angle while the vehicle is maintaining a straight course. Such a roll angle might be normal if the vehicle is making a turn, but, given the context, the measurement would be considered anomalous and a possible indication of a fault.

Figure 3-2 illustrates a contextual anomaly. The data used in this plot is fabricated, and is meant to represent time-series data with periodic behavior. The anomalous data instance, marked in red, would be considered normal without context, since there are a number of similar observations. In order to identify this point as anomalous, we must account for the time-series aspect of the data.

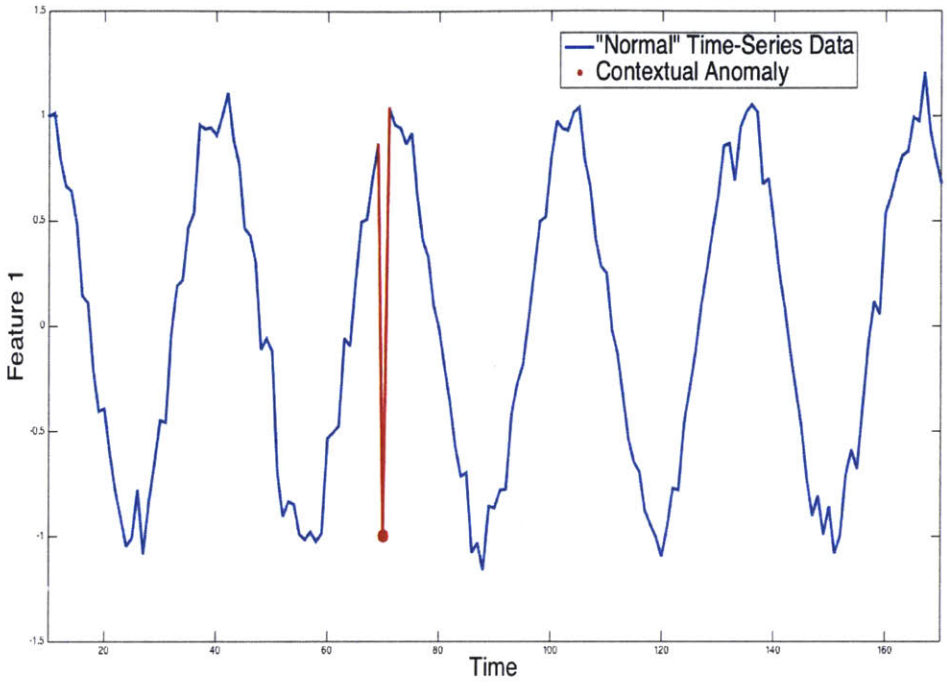


Figure 3-2: Illustration of contextual anomaly in artificially generated periodic data

Collective Anomalies

Lastly, a collective anomaly is a collection of related data instances that is anomalous with respect to the entire data set[9]. The individual data instances may not be considered anomalous on their own. Like contextual anomalies, dealing with collective anomalies is typically more complex, since we have to account for additional information (i.e. the relationships between data instances).

An example of a collective anomaly for UUV data would be the absence of a depth reading for a sequence of consecutive depth measurements. It is not uncommon for a vehicle to lose bottom lock, which refers to the event when a vehicle is unable to obtain sensor readings of the ocean floor. If the UUV loses bottom lock, the depth measurement returns 0. Typically, the UUV will regain bottom lock relatively quickly. In this case, we might consider a sequence of twenty consecutive depth readings of 0 as anomalous. Note that for this example it would be easy to transform certain collective anomalies into point anomalies. Instead of measuring depth readings, we

could instead consider the number of observations since the previous non-zero observation. Under this transformation, a sequence of twenty consecutive depth readings of zero would be a point anomaly.

Figure 3-3 illustrates a collective anomaly for fabricated time-series data. The anomalous subset is a sequence that shows very little variance between consecutive observations.

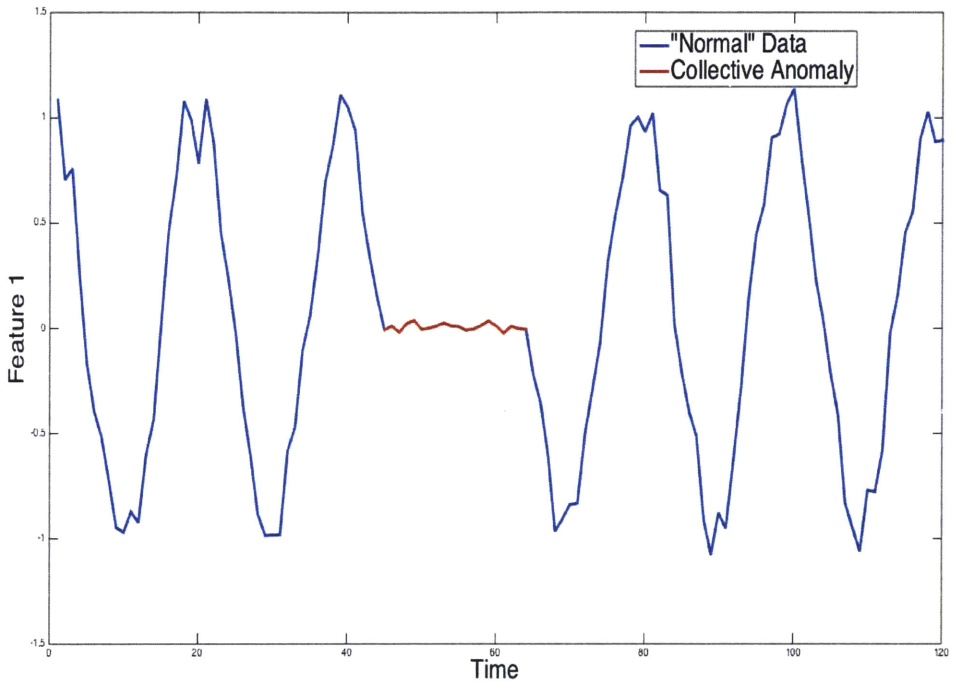


Figure 3-3: Illustration of collective anomaly in artificially generated time-series data

3.2.2 Nature of Data

Another characteristic of an anomaly detection problems is the nature of the available data. We refer to each data point as a *data instance*, and each data instance is made up of one more *variables*, or *attributes*. These variables may be continuous, discrete, binary, or categorical. If data instances have one variable, they are said to be *univariate*. Data instances with two or more variables are *multivariate*.

We must also consider the relationships between data instances. Two data in-

stances, say A and B, are said to be independent if the attributes of A are not affected by the attributes of B, and vice versa. More formally, if A and B are generated by some data generating process, then A and B are independent if

$$Pr(A \cup B) = Pr(A) * Pr(B).$$

That is, the joint probability of data instance A and B both appearing in our data set is equal to the product of the marginal probability of A appearing and the marginal probability of B appearing. Data instances are *dependent* if

$$Pr(A \cup B) \neq Pr(A) * Pr(B).$$

Time-series and sequential data tend to be highly dependent, since neighboring data instances are typically related.

The applicability of anomaly detection techniques is highly dependent on the nature of our data. Anomaly detection techniques may not be applicable for all types of data. For example, nearest neighbor techniques, which will be discussed in the following chapter, require a distance metric defined over the domain of our data set. Distance metrics (e.g. Euclidean distance) are easy to define over continuous data, but do not translate to categorical data. Furthermore, the majority of existing techniques assume that data instances are independent. Identifying anomalies in dependent data sets typically requires more sophisticated algorithms.

3.2.3 Availability of Labeled Data

Anomaly detection problems are also dependent on the availability of labeled training data. A data instance can be labeled as "anomalous" or "normal". Obtaining labeled training data may be difficult or impractical for a number of reasons. Labeling data may be prohibitively expensive, due to time constraints or the size of a data set. Another difficulty may be in obtaining data that covers the range of possible anomalies, particularly if anomalous data instances correspond to rare events. Depending on the

availability of labeled data, there are three approaches to anomaly detection, namely supervised, semi-supervised, and unsupervised anomaly detection.

Supervised Anomaly Detection

Supervised anomaly detection methods require labeled training data from both the "normal" and "anomalous" classes. The goal of supervised anomaly detection is to build a classifier that can distinguish between these two classes. This is also known as *two-class classification*, and is often done by building probability models for each class. With this strategy, future instances are classified by comparing the conditional probabilities that the instance belongs to each class. Another strategy is to fit classifiers directly, without building probabilistic models. The support vector machine is one such classifier, and does not require any assumptions about the data generating process [17].

Utilizing supervised anomaly detection methods can be difficult. As previously stated, obtaining sufficient labeled training data is often infeasible. Even with training data from both classes there is often the issue of data imbalance. This occurs when the training set contains many more instances from the "normal" class, and is often the case for anomaly detection problems[11].

Semi-Supervised Anomaly Detection

Semi-Supervised anomaly detection methods require data from the "normal" class. These methods are more widely applicable than supervised methods, since normal data is often readily available for most applications. A common approach for semi-supervised methods is to develop a probabilistic model for normal behavior, and classify future instances as anomalous if they lie in regions of low probability.

Unsupervised Anomaly Detection

Lastly, *Unsupervised* anomaly detection methods can be used when labeled training data is unavailable. The key assumption for unsupervised methods is that anomalous

data instances appear much less frequently than normal data instances in our training set. As stated in [9], if this assumption does not hold true, then these methods can lead to a very high false-alarm rate. These methods are widely applicable, since they do not require labeled data.

3.2.4 Output

The last characteristic of an anomaly detection problem is the desired output to be reported. Chandola, et. al, list two possible outputs: *anomaly scores* and *labels*[9].

An anomaly score is a numeric value assigned to a data instance that represents the degree to which the instance is considered anomalous. Typically, higher scores are used for anomalous data instances. An example would be if we create a probabilistic model for normal data under a semi-supervised technique. Suppose we estimate a probability density function, $\hat{f}_X$, for normal data. A reasonable anomaly score for a test point would be the inverse of the density function evaluated at that point. That is, for a point, x, in our test data set, we have

$$score(x) = \frac{1}{\hat{f}_X(x)}.$$

An analyst might sort test data by anomaly score, and then select the instances with the highest scores to analyze further.

The alternative is to output a label, either "anomalous" or "normal". This is typically done by selecting a certain threshold for what is considered anomalous. For many techniques, outputting a label is equivalent to setting a predetermined threshold on anomaly scores. For other techniques, such as SVM, the relationship between an anomaly score and a label is not as straightforward.

3.2.5 Characteristics of UUV Anomaly Detection

We now discuss the characteristics of the UUV anomaly detection problem that is of interest in this thesis. These characteristics will allow us to narrow down potential anomaly detection techniques that may be applicable to our data.

As stated in our problem statement in Chapter 1, the goal of this research is to help operators identify faults in UUV systems that are previously unmodeled. Specifically, we are interested in determining if systems are behaving normally over an entire mission. As discussed in Chapter 2, most UUV performance data is time-series, but we will not address the problem of finding anomalous sequences within this time-series data. According to experienced operators, a vehicle will not typically experience an unmodeled fault and then correct itself in the middle of the mission. For example, if a part of the thruster system breaks, it will remain broken for the entire mission. Instead, we will use features of performance data to identify anomalous system behavior over the course of an entire mission. We must select appropriate features that are indicative of system performance

Our problem is thus to find point anomalies, where our data are specific features of performance data. Each point in our data set corresponds to a feature of time-series data from an individual mission. As an example, suppose we are interested in identifying anomalous thruster behavior. UUVs typically collect time series data of thruster ouput and thruster goal over the course of a mission. A feature that we could use is the average of the difference between thruster output and thruster goal. If thruster data is collected from time $t = 1$ to T then the feature, X_i, of mission i is:

$$X_i = \frac{\sum_{t=1}^{T} ThrusterOutput(t) - ThrusterGoal(t)}{T}$$

For this problem, we do not have available labeled data. The UUVs of interest do generate a fault log, which is typically an ad hoc list of faults that are previously modeled. The vehicle is programmed to automatically identify these faults, and an operator will be alerted to them by reading through the fault log after a mission. Since we are interested in identifying unmodeled faults, we do not have the luxury of using these fault logs to obtain labeled data. Furthermore, it would be implausible to obtain sufficient labeled data for the following reasons:

- Unmodeled faults are rare relative to modeled faults and normal performance data.
- Unmodeled faults are not well documented.
- Due to the complexity of UUV systems, there is a wide range of possible faults, which may correspond to a wide range of anomalous data.

Due to the lack of labeled data, we will focus on unsupervised anomaly detection methods. Due to the fact that operators are under a time-constraint when performing post-mission data analysis, we will focus on methods that output a label, rather than an anomaly score. This will save operators the step of analyzing a sorted list of anomaly scores.

In conclusion, our goal is to identify anomalous performance data by using features of time-series data from individual missions. We are interested in methods that output an anomaly label. Due to lack of available labeled data, we will use unsupervised methods, and make the assumption that anomalies in our test set are rare. In the following chapter, we will discuss one-class classification methods that could be appropriate for this task.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

One-Class Classification Methods

Based on the characteristics of the UUV anomaly detection problem, we have determined that one-class classification methods are most suitable for identifying anomalous features of UUV data. In this chapter, we discuss specific one-class classification methods that may be applicable, namely statistical methods, distance-based methods, and the one-class support vector machine.

These methods can be utilized for both semi-supervised and unsupervised anomaly detection. In the semi-supervised case, we have labeled training data from the normal class. In the unsupervised case, we have unlabeled data, but we rely on the assumption that anomalous data instances are rare in our test set. As previously stated, for this research we are assuming that unmodeled faults, which typically correspond anomalous mission performance data, are rare events.

4.1 Statistical Methods

The general strategy with statistical anomaly detection techniques is to estimate some probability distribution using historical data. The estimated distribution is then used to make a decision on future data points. The key assumption for these techniques, as stated in [9], is that "normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model". These methods also rely on the assumption that normal data

instances are independent and identically distributed.

In this section, we will discuss both parametric and non-parametric techniques. The key distinction is that parametric techniques require extra assumptions about our data, namely that the data is generated from a family of probability distributions, and that these distributions have a fixed number of parameters. If these assumptions are correct, then parametric techniques will typically lead to better results. Non-parametric techniques make no assumptions on the underlying distribution of our data.

4.1.1 Parametric Techniques

With parametric techniques we make an assumption about the family of distributions to which the data generating process of interest belongs. Our goal is to estimate a probability density function, $f_X(x;\Theta)$, where x is a data instance and Θ represents the parameters of the density function. In other words, we attempt to estimate a probability density function over x, parameterized by Θ. We estimate Θ in one of two ways.

First, we can estimate Θ by using the *maximum likelihood estimate* (MLE). This is a function of our training data. The idea of MLE is to find the parameters that maximize the likelihood that our data was generated by $f_X(x;\Theta)$. Suppose our training data consists of the instances $x_i \in D$. The maximum likelihood estimate, $\hat{\Theta}(D)_{MLE}$ is given by:

$$\hat{\Theta}(D)_{MLE} = \underset{\Theta}{arg\,max} \prod_{x_i \in D} f_X(x_i;\Theta).$$

Alternatively, we can incorporate some prior knowledge about our data to find the *maximum a posteriori probability* (MAP) estimate for Θ. Suppose we have a prior distribution on our parameters, say $g(\Theta)$. This prior distribution would typically be developed from expert knowledge on the system of interest. From Bayes' theorem, the maximum a posteriori probability estimate, $\hat{\Theta}_{MAP}$ is given by:

$$\hat{\Theta}_{MAP}(D) = \underset{\Theta}{arg\,max} \prod_{x_i \in D} \frac{f(x_i|\Theta)g(\Theta)}{P(x_i)}$$
$$= \underset{\Theta}{arg\,max} \prod_{x_i \in D} f(x_i|\Theta)g(\Theta)$$

Using our estimated parameters, Θ, we can output an anomaly score for a test instance, x_i, corresponding to the inverse of the density function evaluated at x_i. We can label data instances as anomalous by setting a threshold on the anomaly score. Typically, this threshold is determined by estimating of the frequency of anomalies in our data. As an example, we consider Gaussian model based anomaly detection.

The advantage of MAP estimation is that it allows us to incorporate expert knowledge into our probability models. If we have an appropriate prior for Θ, then MAP estimation will likely lead to better results.

Gaussian Model

In this simple example, suppose we have n univariate data instances, $\{x_1, x_2, \ldots, x_n\} = D_{training}$. The assumption in Gaussian model based anomaly detection is that our data is generated by a univariate Gaussian probability density function with parameters $\Theta = \{\mu, \sigma\}$. For this example, we estimate our parameters, $\hat{\Theta} = [\hat{\mu}, \hat{\sigma}]$, using maximum likelihood estimates. These estimates are given by:

$$\hat{\Theta} = [\hat{\mu}, \hat{\sigma}] = \underset{[\mu,\sigma]}{arg\,max} \prod_{x_i \in D} f_X(x_i; [\mu, \sigma]). \tag{4.1}$$

As shown in [18], the values $\hat{\mu}$ and $\hat{\sigma}$ that maximize equation 4.1 the sample mean and sample standard deviation, respectively. That is,

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}{n}}.$$

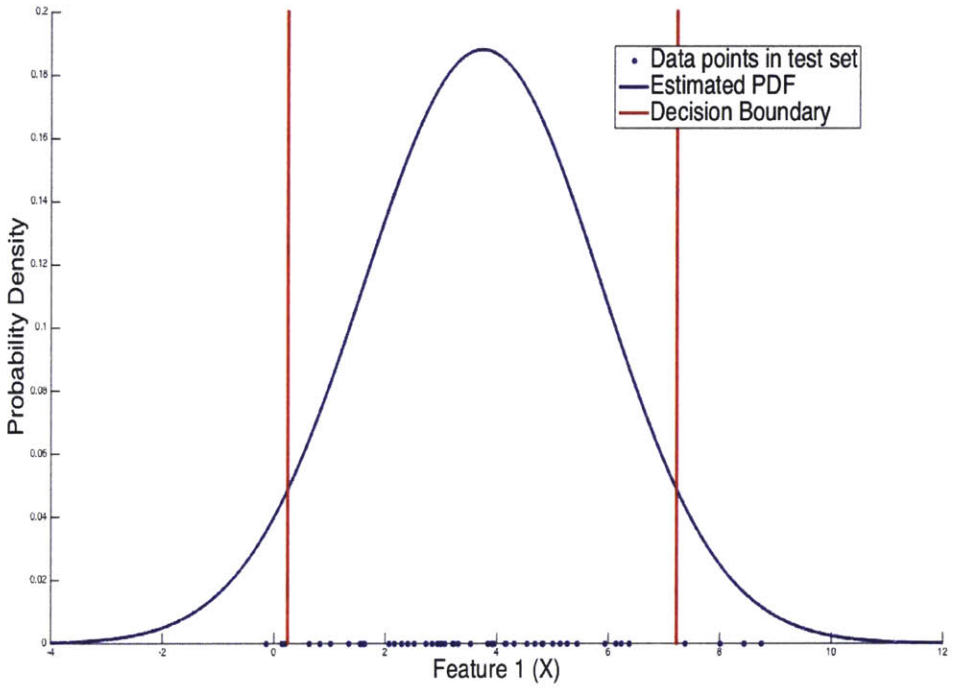


Figure 4-1: Gaussian Model Based Anomaly Detection for Fabricated Data

Figure 4-1 shows an example for fabricated test data generated from a Gaussian distribution with $\mu = 4$, and $\sigma = 2$. We used maximum likelihood estimates to compute $\hat{\mu}$ and $\hat{\sigma}$ from our test data. For this example, we estimated that 10% of our features are anomalous due to unmodeled faults. We thus compute a 90% confidence interval for data points generated by our estimated PDF. Boundaries for our confidence interval are given by $\hat{\mu} \pm z * \hat{\sigma}$, where z is the point where the cumulative density function is equal to $\frac{0.10}{2} = 0.05$, that is $F_{\hat{\Theta}}(z) = 0.05$. Once we have our decision boundary, a test point x_{test} is classified as anomalous if it lies outside of our

computed confidence interval.

4.1.2 Non-parametric Techniques

With non-parametric techniques, we are not required to make an assumption about the underlying distribution of our data. Similar to parametric techniques, we attempt to estimate the distribution of the data generating process, and we classify test instances as anomalous if they fall into regions of low probability. Two non-parametric techniques that were explored in this research are histogram-based anomaly detection, and anomaly detection using kernel density estimation.

Histogram-Based

Histogram-based anomaly detection is the simplest non-parametric technique available. For univariate data, this involves creating a histogram based on training data, and classifying test instances as anomalous if they fall into bins with zero or few training instances. The key parameter in this case is the size of bins in our histogram. If our bin size is too small, we are more likely to have normal instances fall into bins with few or no training instances. This will lead to a high false alarm rate. On the other hand, if the bin size is too large, we are more likely to have anomalous instances fall into bins with a higher frequency of training data, leading to a higher rate of true negatives[9].

One way to choose a bin size for histogram-based methods would be to estimate the number of unmodeled faults in our training set. We could fix the threshold, start with a large bin size, and decrease bin size until the number of anomalies classified in our training set is approximately equal to our estimate of the number of unmodeled faults. Alternatively, we could fix bin size and adjust our threshold until we achieve the same number of faults in our training set.

Kernel Density Estimation

Kernel density estimation, also known as *Parzen Windows Estimation*, is a non-parametric technique for estimating a data generating process. A *kernel*, $K(\cdot)$ is any non-negative function that has mean zero and integrates to one. Two commonly used kernels are the Gaussian kernel (K_G) and the Epanechnikov kernel (K_E). These kernels are defined by the following equations:

$$K_G(x) = \frac{1}{\sqrt{2\pi}} exp(\frac{-x^2}{2})$$
$$K_E(x) = \frac{3}{4}(1 - x^2)\mathbf{1}_{\{|x| \leq 1\}}$$

Given a set of training data, $\{x_1, \ldots, x_n\} = D_{training}$, and kernel, $K(x)$, we estimate the probability density function over x with the following equation:

$$\hat{f}_X(x) = \frac{1}{n * h} \sum_{i=1}^{n} K(\frac{x - x_i}{h}),$$

where h is a smoothing parameter known as the *bandwidth*. We can also include h in our characterization of our kernel function, by letting $K_h(x) = \frac{1}{h} K(x/h)$. If we use K_h, also known as a scaled kernel, our density estimate reduces to the following:

$$\hat{f}_X(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i)$$

In words, if we have n training data instances, our density estimate is the average of n scaled kernel functions, one centered at each of the points in our training set[19]. Figure 4-2 illustrates a KDE approximation for a univariate PDF. The estimated PDF (black) is the sum of scaled kernel functions (blue) centered at our data (red).

Once we have an estimate for a probability density function, the problem of classifying anomalies is identical to that of parametric statistical methods. We classify a test instance, x_{test}, as anomalous if it lies in an area of low probability. In the following chapter, we will explore methods for computing decision boundaries based

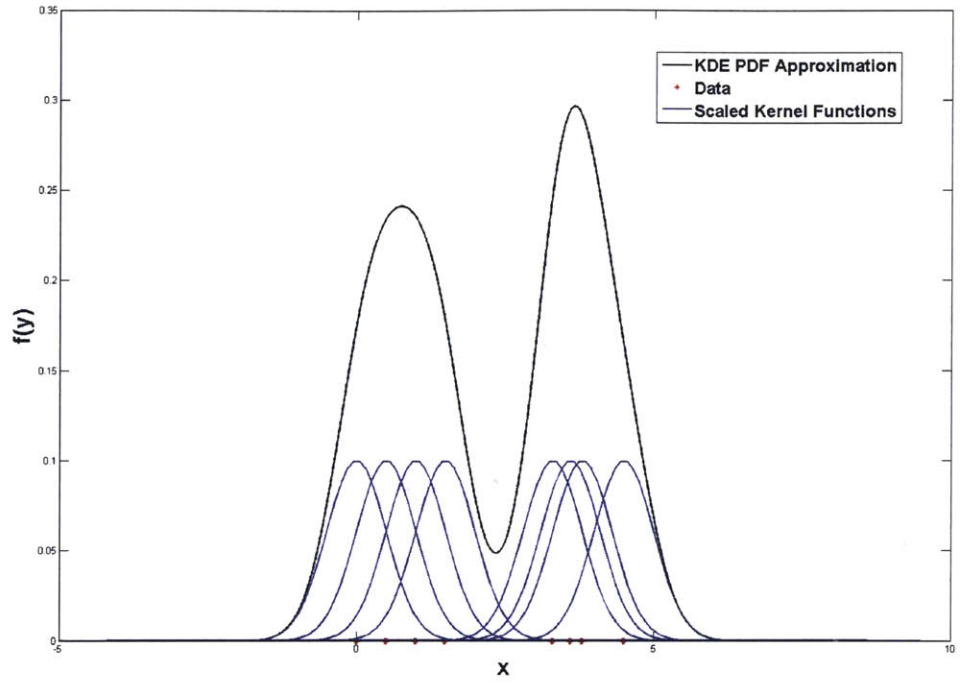


Figure 4-2: Kernel Density Estimate for univariate data

on PDFs estimated using kernel density estimation. We will also explore parameter selection for bandwidth, h.

4.2 Distance Based Methods

Nearest neighbor and clustering algorithms are common in machine learning literature. In this section we will discuss how *nearest neighbors* and *clustering* algorithms have been adapted for the purpose of one-class classification. Unlike statistical methods, these techniques do not involve stochastic modeling.

These techniques require a distance metric to be defined between two data instances. The distance metric used in this research is Euclidean distance, d_e. For instances $x_i, x_j \in D \subset \mathbb{R}^n$, Euclidean distance is defined as

$$d_e(x_i, x_j) = \sqrt{(x_i^1 - x_j^1)^2 + (x_i^2 - x_j^2) + \ldots (x_i^n - x_j^n)}$$
$$= \sqrt{(x_i - x_j) \cdot (x_i - x_j)}.$$

Another commonly used distance metric is Mahalanobis, or statistical distance, which takes into account correlation between variables. This is a useful distance metric if we believe that our variables are highly correlated.[18].

4.2.1 k-Nearest Neighbors

As stated in [9], k-nearest neighbor anomaly detection relies on the assumption that "Normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors". Incorporating k-nearest neighbors for one-class anomaly detection requires the following steps[22]:

1. Store all data instances from training set, $D_{training}$.

2. Select parameter k and distance metric, $d(x_i, x_j)$.

3. For each $x \in D_{training}$, identify the k-nearest neighbors in $D_{training}$. We will denote the set of k-nearest neighbors of x_i as $N(x_i)$.

4. Compute an anomaly score for each instance $x \in D_{training}$. The anomaly score for instance x_i is given by the averaged distance from x_i to its k-nearest neighbors:
$$anomaly\ score(x_i) = \frac{1}{k} \sum_{x_j \in N(x_i)} d(x_i, x_j)$$

5. Set a threshold, denoted $threshold_{knn}$, on anomaly scores. This is determined by our belief of the frequency of faults in our test set. If we believe that a fraction, p, of n missions in our test set contain unmodeled faults, then we select $threshold_{kNN}$ to be the $(p * n)^{th}$ highest anomaly score. If we have

confirmed that our test set contains no unmodeled faults then we would select $threshold_{kNN} = max\{anomaly\,score(x_i) : x_i \in D_{test}\}$.

6. For a test instance, $x_{test} \in D_{test}$, identify the k-nearest neighbors within the training set.

7. Compute the anomaly score for x_{test} as in step 4:

$$anomaly\,score(x_{test}) = \frac{1}{k} \sum_{x_j \in N(x_{test})} d(x_{test}, x_j).$$

8. If $anomaly\,score(x_{test}) > threshold_{kNN}$, classify instance as anomalous. Otherwise, classify as normal.

4.2.2 Local Outlier Factor (LOF)

One shortcoming of many distance based techniques, including k-nearest neighbors, is that they only account for global density of our data set. For more complex real-world data sets, our data may contain regions of higher variance. In this case, we would also like to take into account local density when identifying point anomalies.

Consider figure 4-3, which contains two clusters of normal data, as well as a point anomaly. The two clusters contain the same amount of data instances, but the bottom left cluster has a much higher local density. The point anomaly would typically not be identified by conventional k-nearest neighbors or clustering techniques, since it is still closer to its nearest neighbors than instances in the top right cluster.

In 1999, Breunig et. al, developed a distance based anomaly detection technique, similar to k-nearest neighbors, that takes into account local density[5]. This technique is known as *local outlier factor* (LOF) anomaly detection. Implementing LOF for one-class anomaly detection requires the following steps:

1. Store all data instances from training set, $D_{training}$.

2. Select parameter, k, and distance metric, $d(x_i, x_j)$.

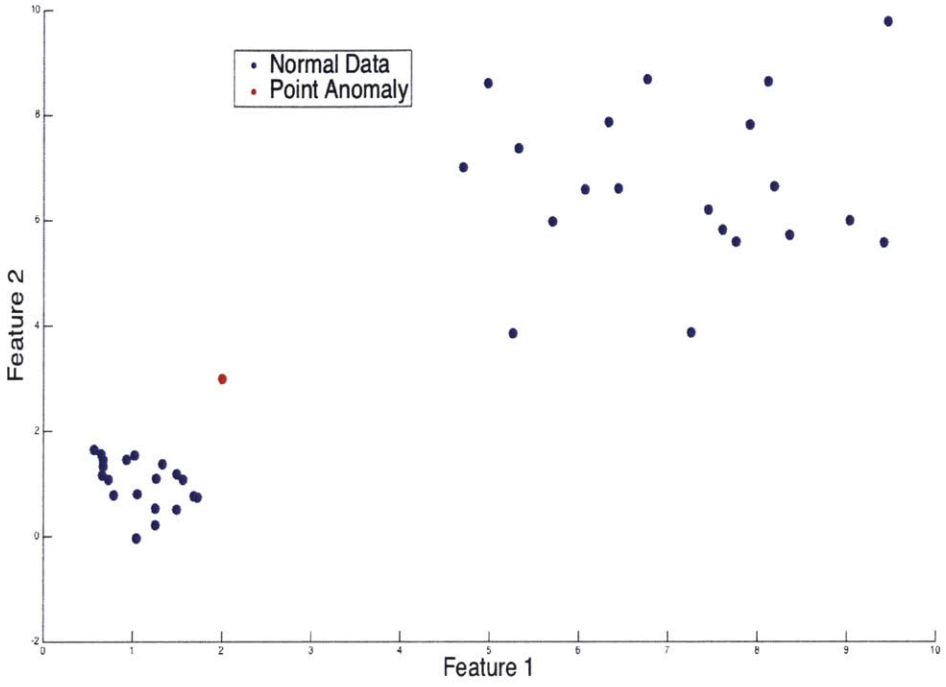


Figure 4-3: Illustration of data set suitable for Local Outlier Factor Anomaly Detection

3. For each $x \in D_{training}$, identify the k-nearest neighbors in $D_{training}$. We will denote the set of k-nearest neighbors of x_i as $N(x_i)$.

4. For each $x_i \in D_{training}$, compute the local density of x_i, which is given by:

$$\begin{aligned} \textit{Local Density}(x_i) &= \frac{||N(x_i)||}{\text{radius of smallest hypersphere centered at } x_i \text{ containing } N(x_i)} \\ &= \frac{k}{max\left\{d(x_i, x_j) | x_j \in N(x_i)\right\}}. \end{aligned}$$

5. For each $x_i \in D_{training}$, compute the local outlier factor of x_i. This value will serve as an anomaly score, and is given by:

$$LOF(x_i) = \frac{\text{average local density of k-nearest neighbors of } x_i}{\text{local density}(x_i)}$$
$$= \frac{\frac{1}{k} \sum_{x_j \in N(x_i)} \text{local density}(x_j)}{\text{local density}(x_i)}$$

6. Set a threshold, denoted $threshold_{LOF}$, on anomaly scores. Again, this is determined by our belief of the frequency of faults in our test set. If we believe that a fraction, p, of n missions in our test set contain unmodeled faults, then we select $threshold_{LOF}$ to be the $(p * n)^{th}$ highest local outlier factor.

7. For a test instance, $x_{test} \in D_{test}$, compute $LOF(x_{test})$, using $D_{training}$ to identify the k-nearest neighbors of x_{test}.

8. If $LOF(x_{test}) > threshold_{LOF}$, classify instance as anomalous. Otherwise, classify as normal.

4.2.3 Clustering

Clustering is the process of partitioning a data set so that similar objects are grouped together. There is extensive literature available for a number of different clustering techniques. Some common approaches to clustering include[18]:

- Model Based Clustering
- Affinity Propagation
- Spectral Clustering
- Hierarchical Clustering
- Divisive Clustering

We will not go into detail about various clustering algorithms in this thesis. Instead, we will discuss techniques for anomaly detection for data that have already

been clustered. In particular, we assume that our training data set, $D_{training}$, has been partitioned into clusters, and our goal is to classify a test instance, x_{test}.

Chandola, et al. distinguish between three different categories of cluster-based anomaly detection. These distinctions are based on assumptions about anomalous data instances. The assumptions of the three categories are as follows[9]

- **Category 1:** Normal data instances belong to clusters, whereas anomalies do not belong to clusters, based on a given clustering algorithm that does not require all data instances to belong to a cluster.

- **Category 2:** Normal data instances lie close to the nearest cluster centroid, whereas anomalous data instances do not.

- **Category 3:** Normal data instances belong to large, dense clusters, while anomalies belong to small and sparse clusters.

The first category requires a clustering algorithms that does not force all data instances to belong to a cluster. The main idea is to use such a clustering algorithm on the training data set together with the test data point (i.e. use clustering algorithm on $D_{training} \cup x_{test}$). If x_{test} does not belong to a cluster (or is the only point in a cluster), then classify as anomalous. Otherwise, x_{test} is classified as normal. An example of such an algorithm would be hierarchical agglomerative clustering. This technique begins with all data instances as individual clusters, then merges data points one at a time based on a certain distance metric[18]. This category is not well-suited for our problem, since it requires that training data is clustered along with test data. We would prefer a technique in which clustering is done prior to having access to x_{test}, in order to save on computation time.

The second category of cluster based anomaly detection requires all data to belong to a cluster. The typical approach would be to partition $D_{training}$ into clusters, and then classify x_{test} as anomalous if it outside a certain threshold distance from the closest cluster centroid. This is a promising approach if we believe that our performance data features form distinct clusters.

The third and final category of cluster based anomaly detection also requires an algorithm that assigns each data point to a cluster. The basic approach in this category is to classify data instances as anomalous if they belong to clusters whose size or density is below a certain threshold. Similar to the second category, we would cluster data in $D_{training}$ first. We would then classify certain clusters as anomalous if they lie outside of a threshold for size or density. Lastly, we would determine which cluster x_{test} would belong, and classify as anomalous or normal based on the cluster to which it belongs. An example of such a technique is *Cluster-Based Local Outlier Factor*, proposed by He, et al., in 2003[12]. The technique incorporates both the size of the cluster, and the local density of a data instance when computing an anomaly score.

k-means clustering

We now outline a simple implementation of anomaly detection using k-means clustering. This implementation falls into the second category of cluster based anomaly detection. The steps are as follows:

1. Use k-means clustering to identify partition $D_{training}$ into k clusters, denoted $C_1, C_2, \ldots C_k$.

2. For each cluster, $C_j \in \{C_1, C_2, \ldots, C_k\}$ compute cluster centroid, μ_j, given by

$$\mu_j = \sum_{x_i \in C_j} \frac{x_i}{|C_j|}$$

3. For each $x_i \in D_{training}$, compute distance to closest centroid. This will be the anomaly score for data point x_i.

4. Set a threshold on anomaly scores based on the belief in the fraction of unmodeled faults in our training set. We will denote this value as $threshold_{cluster}$.

5. For test data instance, x_{test}, compute the distance to the closest cluster cen-

troid. If distance is greater than $threshold_{cluster}$, classify x_{test} as anomalous. Otherwise, classify x_{test} as normal.

4.3 One-Class SVM

The one-class support vector machine (SVM) is an approach that attempts to fit a classifier directly to the training data, without any probabilistic assumptions or decision theory applications. The assumption for this approach is that "a classifier that can distinguish between normal and anomalous classes can be learnt in the given feature space"[9]. In this section we will briefly describe the more traditional two-class support vector machine, and explain how it has been adapted for the one-class classification problem.

Two-Class SVM

The support vector machine is most commonly used for two-class classification. This requires that we have labeled training data from two distinct classes (e.g. normal class and anomalous class). The main idea is to find a separating hyperplane that maximizes the margin, which is the distance between the hyperplane and the closest data points. Suppose we have training, $D_{training} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. In this case, $x_i \in D_{training}$ is a training instance and y_i is a class label (e.g. -1 = normal, 1 = anomalous). Our goal is to find the optimal hyperplane, given by $w * x + w_0 = 0$, where $w \in \mathbb{R}^d$, and $w_0 \in \mathbb{R}$. We can find an optimal hyperplane by solving the following quadratic program:

$$\text{minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \text{ subject to:}$$

$$y_i(w \cdot x_i + w_0) \geq 1 - \xi_i \text{ for } i = 1, \ldots, n$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, n$$

The inclusion of C and slack variables ξ_i allows some points to be inside the margin. If we increase C, we force more points to be on the correct side of the margin, at the cost of having a smaller margin[18].

One interesting and useful result with support vector machines, is that the above optimization problem can be rewritten in terms of dot products of our data instances. This allows us to easily implement kernel functions. A *kernel function*, typically denoted $\Phi(x)$, implicitly maps our data instances into a higher dimensional inner product space. Note that this is different than the kernel that we defined for kernel density estimation. In this case, using a kernel function allows us to find an optimal hyperplane in a higher dimensional space, which will appear as non-linear classification boundary in $\mathbb{R}^d$.

A commonly used kernel function is the radial basis kernel function, or RBF kernel. The RBF kernel is given by

$$\Phi(x_i) \cdot \Phi(x_j) = K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

The RBF kernel is a similarity measure between data instances x_i and x_j. By using this kernel, we are implicitly mapping our data instances into an infinite dimensional feature space, i.e. $\Phi(x_i) \in \mathbb{R}^\infty$. The fundamental concept is that we do not need to calculate $\Phi(x_i)$, since we are only concerned with dot products between vectors, which is given by $K(x_i, x_j)$[18].

One-Class SVM (Schölkopf et al.)

In 1999, Schölkopf et al., adapted the support vector machine for one-class classification[20]. The main idea of their technique is to separate a majority of the training data from the origin in some high-dimensional feature space by utilizing kernel functions. The result gives a non-linear decision boundary in the original feature space that encapsulates the majority of the training data. For training data instances, $x_i \in D_{training}$, and kernel function $\Phi(x)$, the optimal hyperplane is found in feature space F by solving the following quadratic program for decision variables ρ, ξ, w, and w_0:

$$\text{minimize } \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \text{ subject to:}$$

$$(w \cdot \Phi(x_i) + w_0) \geq \rho - \xi_i \text{ for } i = 1, \ldots, n$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, n$$

Recall that for two-class SVM, the parameter C determined how strict we were in requiring data instances to be on the correct side of the margin. For this implementation of one-class SVM, the parameter ν sets an upper bound on the fraction of data instances that lie outside of the decision boundary[20].

In order to implement one-class SVM for UUV performance data, we would first select an appropriate kernel function (e.g. radial basis kernel). We would then solve the dual form of the optimization problem above for dual variables, α_i, and data instances $x_i \in D_{training}$. A test data instance, x_{test}, would be classified by using the resulting decision function:

$$f(x_{test}) = sgn((w * \phi(x_{test}) + w_0) - \rho) = sgn(\sum_{i=1}^{n}\alpha_i K(x_i, x_{test}) - \rho)$$

If $f(x_{test}) = -1$, then we classify x_{test} as anomalous. Otherwise, we classify x_{test} as normal[20].

One-Class SVM (Tax & Duin)

In 2004, Tax and Duin developed another version of one-class SVM called Support Vector Data Description (SVDD)[21]. Rather than finding an optimal hyperplane, Tax and Duin take the approach of finding an optimal hypersphere that encapsulates the region of normal data. The hypersphere is characterized by center, a, and radius, R. For training data instances $x_i \in D_{training}$, an optimal hypersphere can be found by solving the quadratic program for decision variables R and a:

$$\text{minimize } R^2 + C\sum_{i=1}^{n}\xi_i \text{ subject to:}$$

$$||x_i - a||^2 \leq R^2 + \xi_i \text{ for } i = 1, \ldots, n$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, n$$

Similar to two-class SVM, the introduction of C and slack variables ξ_i allow us to have data instances in our training set that lie outside of the hypersphere. For small values of C, we will generally have more points outside of the decision boundary.

In order to classify a test data instance, x_{test}, we only need to compute the distance from x_{test} to the center of the hypersphere. If $d(x_{test}, a) \geq R$, then we classify x_{test} as anomalous. Otherwise, we classify x_{test} as normal. In [21], Tax and Duin also show how to implement the radial basis kernel function, which allows to find a decision boundary in our original feature space that is not necessarily a hypersphere[21].

4.4 Discussion

The focus of this section has been to explore one-class classification methods that might be applicable to the problem of finding point anomalies among features of UUV performance data. Each method shown above has certain positive and negative characteristics. We now discuss these pros and cons and narrow down our focus to two methods: Kernel Density Estimation and Local Outlier Factor.

First, parametric statistical methods allow us to make stronger predictions, but only if certain assumptions about our data are correct. The key assumption that we are required to make is that our data is generated from a certain family of distributions. If our assumption about the family of distributions is off, then our predictions will suffer. Based on the available UUV data and features that are of interest, we have determined that making an assumption about the family of distributions is not preferable.

Non-parametric methods give us an alternative way to estimate the data gener-

ating process without making an assumption about the underlying distribution. For this reason, we have determined that anomaly detection using kernel density estimation is a promising technique. Estimating a probability distribution for our data is intuitive, and also allows for useful visualizations for univariate and bivariate data. Furthermore, having an estimate for a density function allows us to easily compute decision boundaries (shown in Chapter 5). One drawback is that we are required to specify a bandwidth parameter, h, beforehand. Anomaly detection with KDE will be the primary focus of our experiments in Chapter 6.

Distance-based methods are also promising. Conventional k-nearest neighbors techniques are very similar to kernel density estimation, in that anomalies will be identified if they lie far from their nearest neighbors among training data. Nearest neighbor methods, however, do not have the advantage of generating a probability distribution, which, in our opinion, makes them less interpretable. Local outlier factor is an interesting technique, since it takes into account the local density of our data set. We have noticed varying local densities in some of our data sets, and we have thus chosen to implement LOF in our experiments in Chapter 6. One drawback of LOF is having to choose parameters k and anomaly threshold. Unlike KDE, where we have heuristics to help us choose the bandwidth parameter, for LOF it is difficult to decide on a parameter k beforehand. Another drawback is that LOF, in comparison to KDE, is much more of a "black box", meaning that the generated classifier may be less interpretable to operators. The last distance based technique, clustering, is not ideal, since we have not found that our data sets contain unique clusters.

Lastly, the one-class support vector machine allows us to fit a classifier without any probabilistic assumptions. The method can be made robust to anomalies in our test set by introducing slack variables into the optimization problem. Also, both techniques that were discussed can be enhanced through the implementation of kernel functions. The primary drawback of the one-class SVM is the interpretability of the generated models. Using kernel functions is a neat trick, but we would also like a method that can be easily understood by operators. Also, one-class SVM requires several parameters (i.e. C, ν, $K(\cdot,\cdot)$) that must be chosen beforehand.

In conclusion, we have determined that kernel density estimation and local outlier factor provide the most upside for our given problem. These methods will be implemented for UUV data, and we will compare their performance. First, we will explore KDE further by looking at both heuristics for bandwidth selection and a method for computing decision boundaries.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Kernel Density Estimation: Parameter Selection & Decision Boundaries

We have determined that anomaly detection using kernel density estimation is a suitable approach for attempting to identify point anomalies among features of UUV performance data. In this chapter, we discuss methods for choosing our bandwidth parameter, and we also discuss ways to compute decision boundaries for our classifier based on our estimated probability density function.

Given that we have an appropriate feature space, there are primarily three challenges in implementing KDE for anomaly detection. The first is choosing an appropriate kernel function, $K(\cdot)$. The second is selecting a bandwidth parameter, h, for our kernel function. The last challenge is choosing an appropriate threshold for which to distinguish between normal and anomalous data instances. For this research, we will use the Gaussian kernel function described in Chapter 4. This is a commonly used kernel, and allows us to use the heuristic described in the following section for choosing a bandwidth parameter. Our discussion of bandwidth selection comes primarily from *Applied Smoothing Techniques for Data Analysis* by Bowman and Azzalini [4]. For the remainder of this chapter, we will denote the true density function as $f(\cdot)$, and our estimated density function as $\hat{f}(\cdot)$

5.1 Parameter Selection

Recall that for kernel density estimation, our goal is to estimate the probability density function for a data generating process. For data, D_{test}, and kernel function, $K(x)$, our kernel density estimate, $\hat{f}(\cdot)$, is given by

$$\begin{aligned}\hat{f}(x) &= \frac{1}{n}\sum_{x_i \in D_{test}} K_h(x - x_i) \\ &= \frac{1}{nh}\sum_{x_i \in D_{test}} K(\frac{x - x_i}{h}),\end{aligned}$$

where $K_h(x) = \frac{1}{h}K(x/h)$ is the scaled kernel function, and h is the bandwidth parameter. Higher values of h give us wider kernel functions, resulting in estimated PDFs that are more smooth and less "peaky".

Selecting too large a bandwidth can lead to oversmoothing, and our resulting estimated density function will underestimate f at peaks in the true density and overestimate at points of low true density. This will likely result in regions of our feature space being labeled as normal that should be considered anomalous. Conversely, having too small a bandwidth will result in an estimated density with many peaks. Our resulting classifier may not fully capture regions of normality.

For our KDE implementations, we will use a heuristic developed by Bowman and Azzalini for computing a bandwidth parameter based on training data. The goal of this heuristic is to minimize the mean integrated squared error (MISE) between $f(\cdot)$ and $\hat{f}(\cdot)$. This is a measure of how effective $\hat{f}(\cdot)$ is in estimating $f(\cdot)$. The MISE for the univariate case is given by the following equation[4]:

$$MISE(\hat{f}) = \mathbb{E}\Big[\int (\hat{f}(x) - f(x))^2 dx\Big] \tag{5.1}$$

$$= \int \Big[\mathbb{E}[\hat{f}(x)] - f(x)\Big]^2 dx + \int \mathrm{var}\Big(\hat{f}(x)\Big) dx \tag{5.2}$$

The expected value of $\hat{f}(x)$ is given by the following:

$$\mathbb{E}[\hat{f}(x)] = \int K_h(x-z)f(z)dz$$

This is the convolution of the true density function, $f(\cdot)$, with the scaled kernel function, K_h. By taking a Taylor series expansion of $\mathbb{E}[\hat{f}(y)]$, we get the following approximation,

$$\mathbb{E}[\hat{f}(x)] \approx f(h) + \frac{h^2}{2}\sigma_K^2 f''(x), \tag{5.3}$$

where σ_K^2 denotes the variance of the kernel function. For our purposes, we will choose a Gaussian kernel with unit variance, so σ_K^2 will reduce to 1. Through another Taylor series expansion, Azzalini and Bowman show that the variance of the estimated density can be approximated by

$$var\Big(\hat{f}(x)\Big) \approx \frac{1}{nh}f(x)\alpha(K), \tag{5.4}$$

where $\alpha(K) = \int K^2(z)dz$. An important result of 5.4 is that the variance of our estimated density is inversely proportional to the sample size, n, of our training set.

By plugging in 5.3 and 5.4 into equation 5.1, we can approximate the mean integrated squared error by the following equation:

$$MISE(\hat{f}) \approx \frac{1}{4}h^4 \int f''(x)^2 dx + \frac{1}{nh}\alpha(K)$$

Using this approximation for the MISE, Azzalini and Bowman show that the bandwidth parameter that minimizes the MISE (for $\sigma_K^2 = 1$) in the asymptotic sense is

$$h_{opt} = \left(\frac{\alpha(K)}{\beta(f)n} \right)^{\frac{1}{5}}, \tag{5.5}$$

where $\beta(f) = \int f''(x)^2 dx$[4]. As pointed out in [4], the expression for h_{opt} given in equation 5.5 is not immediately applicable, since it depends on the unknown distribution, $f(\cdot)$, that we are trying to estimate. The heuristic given by Azzalini and Bowman in [4] is to use the bandwidth, h, that minimizes MISE *given that* $f(\cdot)$ *is Gaussian.* This clearly goes against our approach of attempting to estimate the true density using non-parametric methods. However, this heuristic does give us an easily computable, and commonly used estimate for the bandwidth parameter. For the univariate case, the optimal bandwidth is given by,

$$h_{opt} = \left(\frac{4}{3 * n} \right)^{\frac{1}{5}} \sigma,$$

where σ is the standard deviation of our true density, which can be approximated by a sample estimate. More generally, if our feature space is of dimension d, then the optimal bandwidth parameter in dimension i is given by:

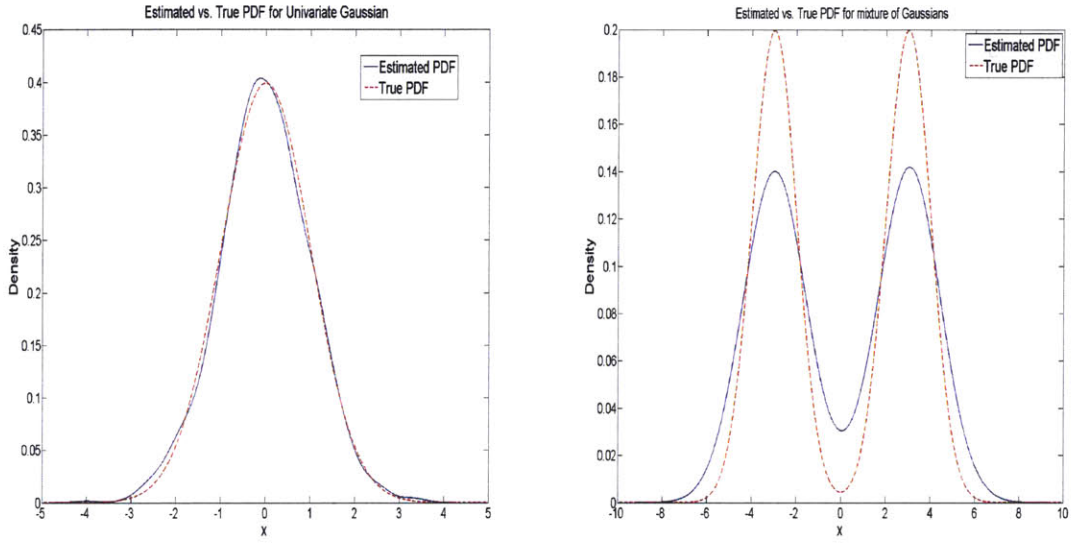
$$h_i = \left(\frac{4}{(p+2) * n} \right)^{\frac{1}{p+4}} \sigma^{(i)}$$

The value $\sigma^{(i)}$ is the standard deviation in dimension i. This can be replaced by a sample estimate for standard deviation. For our models, we will use the median absolute deviation estimator[14]:

$$\hat{\sigma}^{(i)} = \text{median}(|x_j^{(i)} - \hat{\mu}^{(i)}|)/0.6745$$

We will use this heuristic for computing our bandwidth parameter in our experimentation in Chapter 6. One consequence of this heuristic is that it tends to induce oversmoothing when the true distribution is non-Gaussian. Figure 5-1 demonstrates

this consequence. In figure 5-1 (a), the true distribution is Gaussian, and our estimated distribution (blue) provides a good approximation for the true distribution (red). In (b), the true distribution is a mixture of Gaussians, and our KDE approximation overestimates $f(\cdot)$ at troughs, and underestimates it at peaks in the true distribution.



(a) (b)

Figure 5-1: Kernel Density Estimate using Bowman & Azzalini Heuristic Bandwidth for (a) Gaussian and (b) Mixture of Gaussians

5.2 Computing Decision Boundaries

We now look at computing a decision boundary that distinguishes between normal and anomalous regions of our feature space, based on our KDE model. As stated in chapter 4, our assumption is that anomalies appear in regions of low probability in our stochastic model.

Our goal is to find the regions of our feature space that contain the highest density. The threshold, α, will correspond to the integral of our estimated PDF over the anomalous region. Hence, $1 - \alpha$ will be the integral of the estimated PDF over the normal region. We will denote the normal region as N. We will choose α based

on our prior belief on the likelihood of an unmodeled fault. The idea is that if $\mathbb{P}(\text{Unmodeled Fault}) = \alpha$, then roughly $1 - \alpha\%$ of our data will in the normal region. We would like to find region N that corresponds region of our feature space with highest density that contains $1 - \alpha\%$ of the density. This also depends on the assumption that unmodeled faults will not appears as dense clusters in our feature space. If this assumption does not hold, then we may incorrectly classify anomalous regions. More formally, our goal is to find $N \subset \mathbb{R}^d$, where d is the dimension of our feature space, such that

$$\int_{x \in N} \hat{f}(x)dx = 1 - \alpha, \tag{5.6}$$

and

$$\hat{f}(x_1) > \hat{f}(x_2) \;\; \forall x_1 \in N, x_2 \in N^C.$$

Since $\hat{f}(x)$ is the normalized sum of many PDFs (i.e. kernel functions), evaluating the integral in 5.6 and solving for N can be computationally expensive. Rather than attempting to solve this problem directly, we will instead discretize our estimated PDF and use numerical integration to find approximations for 5.6.

In order to find our decision boundary, we must first find a threshold on the value of our PDF. We will denote this threshold as η. Our goal is to find the value η, such that 5.6 holds, where $N = \{x : \hat{f}(x) > \eta$. Algorithm 1 below shows the pseudocode for computing η.

We compute the numerical approximation of the integral in Algorithm 1 using the trapezoidal method. Rather than finding the limits of integration explicitly, we can also set $\hat{f}(x) = 0$ for all x such that $\hat{f}(x) < \eta$ at each iteration. Then we can simply integrate over the entire PDF. This is equivalent to solving for the numerical approximation of $\int_{x \in N} \hat{f}(x)dx$, but saves us the step of finding the limits of integration.

We now illustrate the implementation of algorithm 1 by finding decision bound-

input : Discretized estimated PDF ($\hat{f}(x)$), Δ_η, α
output: Value of PDF at decision boundary, denoted η

$\eta = 0$;
Integral = 1;
while *Integral* $> 1 - \alpha$ **do**
    $\eta = \eta + \Delta_\eta$
    $N = \{x : \hat{f}(x) > \eta\}$
    Integral = Numerical approximation of $\int_{x \in N} \hat{f}(x)dx$
end

Algorithm 1: Pseudocode for computing value of PDF at decision boundary

aries for fabricated univariate and bivariate data. Figure 5-2 illustrates a KDE approximation for a univariate standard Gaussian distribution. The KDE model was computed using 100 random Gaussian data points. We have computed the PDF threshold, η, shown by the horizontal black line, corresponding to $\alpha = 0.05$. The decision boundaries are shown in red. Computing this decision boundary is analogous to finding a 95% confidence interval on a Gaussian distribution with mean 0 and standard deviation 1. In our case, we find the decision boundary by estimating the PDF and then using algorithm 1 to find the PDF threshold. As shown in figure 5-2, our normal region is approximately the interval $[-1.96, 1.96]$, which corresponds to the z-scores for a 95% confidence interval. This is an indication that our algorithm for computing the decision boundary is behaving as expected.

Figure 5-3 (a) shows a KDE approximation for a bivariate PDF. The true PDF is a mixture of Gaussians. In figure 5-3 (b), we have computed the PDF threshold corresponding to $\alpha = 0.9$. The blue points are data instances used to compute the KDE approximation of the true PDF. The decision boundary (green) is shown by a contour plot with a contour line at our PDF threshold. This illustrations shows that we can effectively find disjoint normal regions.

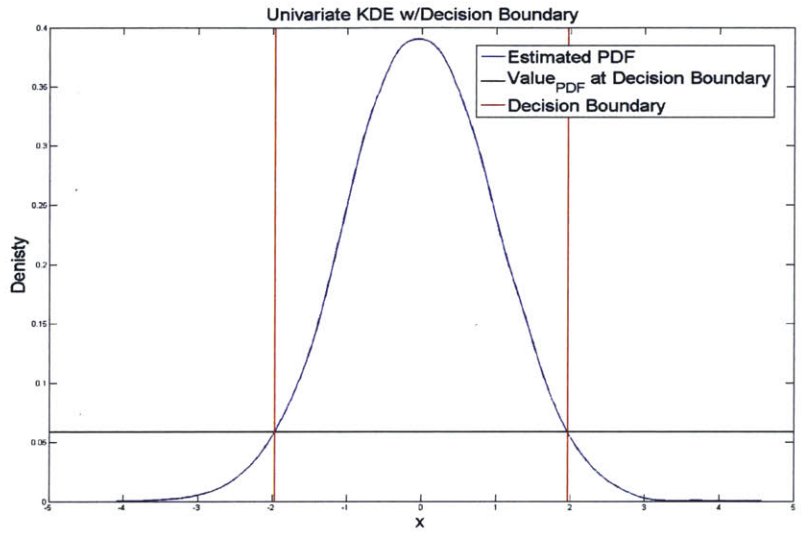
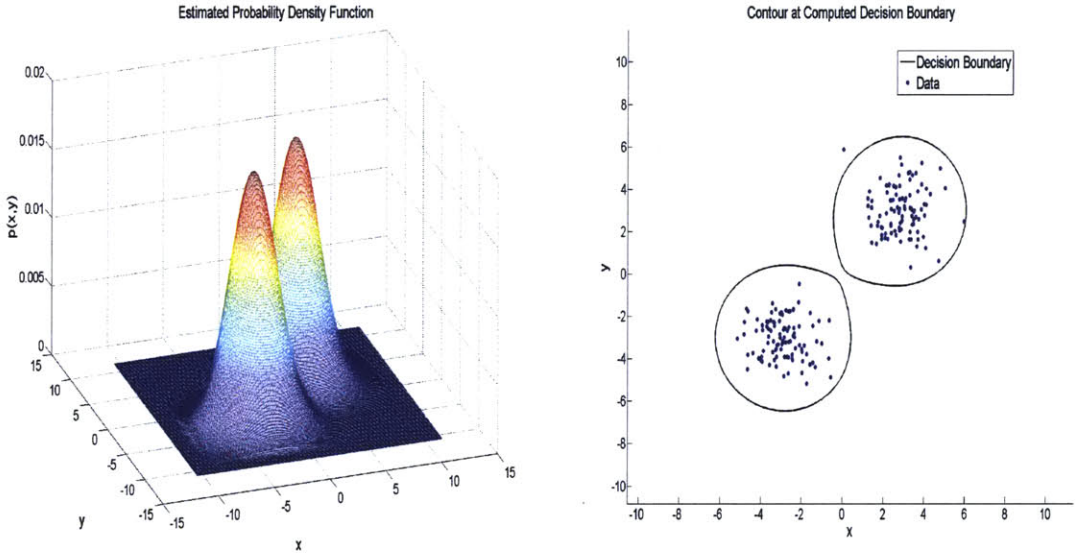


Figure 5-2: KDE Approximation and Decision Boundary for Univariate Gaussian



(a) (b)

Figure 5-3: Illustration of (a) KDE approximation for Multivariate PDF (b) Decision Boundary for Multivariate PDF

5.3 Discussion

In this section, we have discussed parameter selection for implementing KDE anomaly detection in practice. We have also shown how to compute decision boundaries in order to distinguish between normal and anomalous regions of our feature space.

We have chosen to use the heuristic given by Azzalini and Bowman in [4] for choosing our bandwidth parameter. This heuristic allows us to easily estimate the bandwidth that minimizes the mean integrated squared error between our estimated and true distributions. It is important to note, however, that this heuristic can lead to oversmoothing if our data has a distribution that is clearly non-Gaussian.

We have also shown how to compute decision boundaries for our KDE anomaly detection classifier. Note that we can implement the algorithm given in section 5.2 for multivariate data of dimension greater than two. For univariate and bivariate data, however, being able to easily plot a decision boundary may be useful for operators. It will allow operators to visualize normal regions vs. anomalous regions, which may provide useful insight into the systems of interest.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Experimentation

In this section, we describe anomaly detection experiments that were performed with REMUS and MARV data. As discussed in chapter 4, the techniques that we will utilize are kernel density estimation and local outlier factor. Our goal in this section is to demonstrate implementations for these techniques, and to assess performance. For these experiments, we have selected bivariate features in order to better illustrate our models. The same methodology can be used for higher dimensional feature spaces.

6.1 Model Performance

One difficulty when working with unlabeled data is figuring out ways to assess the performance of our algorithms. For supervised techniques, assessing performance is relatively straight-forward since class labels can be compared to anomaly outputs given by our anomaly detection algorithm. Since we do not have class labels readily available, the problem is a little more complex.

Ultimately, we decided to go through a subset of the data and generate our own labels. Our goal is to perform the same analysis that an operator would perform between missions, but only for a particular system. This was done by analyzing time-series data from the system of interest, and incorporating our own knowledge about the system. For each mission, we labeled the system as "normal" or "faulty" based on relevant time-series data. For missions that were labeled faulty, we checked the

fault log to determine if the irregular data was caused by a known fault. If the fault log did not reveal a known a fault, the system for that mission was labeled as an unmodeled fault.

We will further discuss our methods for labeling missions later on in this chapter, since the methods are unique to each system. Note that we are not using labeled data to train our models, for reasons discussed in Chapter 3. We have only generated labeled data by hand for specific UUV systems in order to assess performance.

Given labeled data, we are now able to use standard performance measures, such as accuracy, recall, and precision. Suppose we have a label y for each mission, where

$$y_i = \begin{cases} 1 & \text{if mission } i \text{ is "faulty"} \\ 0 & \text{if mission } i \text{ is "normal"} \end{cases}.$$

Furthermore, suppose our anomaly detection algorithm generates a classifier $\hat{f}(x_i) = \hat{y}_i$, where $\hat{y}_i$ is the output anomaly label for data instance x_i. As in [18] we define four types of decision classifications based on y and $\hat{y}$, which are given in table 6.1. The value N_+ is the total number of faulty missions in our data set, and $\hat{N}_+$ is the

	$y = 1$	$y = 0$	$\sum$
$\hat{y} = 1$	**True Positives (TP)**	**False Positives (FP)**	$\hat{N}_+$
$\hat{y} = 0$	**False Negatives (FN)**	**True Negatives (TN)**	$\hat{N}_-$
$\sum$	N_+	N_-	N

Table 6.1: Decision Classification Table

total number of predicted anomalies as classified by $\hat{f}(\cdot)$. The corresponding N_- and $\hat{N}_-$ values give the number of normal missions and the number of predicted normal missions, respectively. The performance measures that are used are given by the following equations [18]:

$$Recall = \frac{TP}{N_+}$$
$$Precision = \frac{TP}{\hat{N}_+}$$
$$Accuracy = \frac{TN + TP}{N}$$
$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

We can also interpret these performance measures as probabilities. The recall corresponds to the probability that a randomly selected data point in the fault class is correctly identified as anomalous in our model. The precision is the probability that a randomly selected point that is classified as anomalous does indeed correspond to a fault. The accuracy is simply the probability that a randomly selected point is correctly classified by $\hat{f}(\cdot)$.

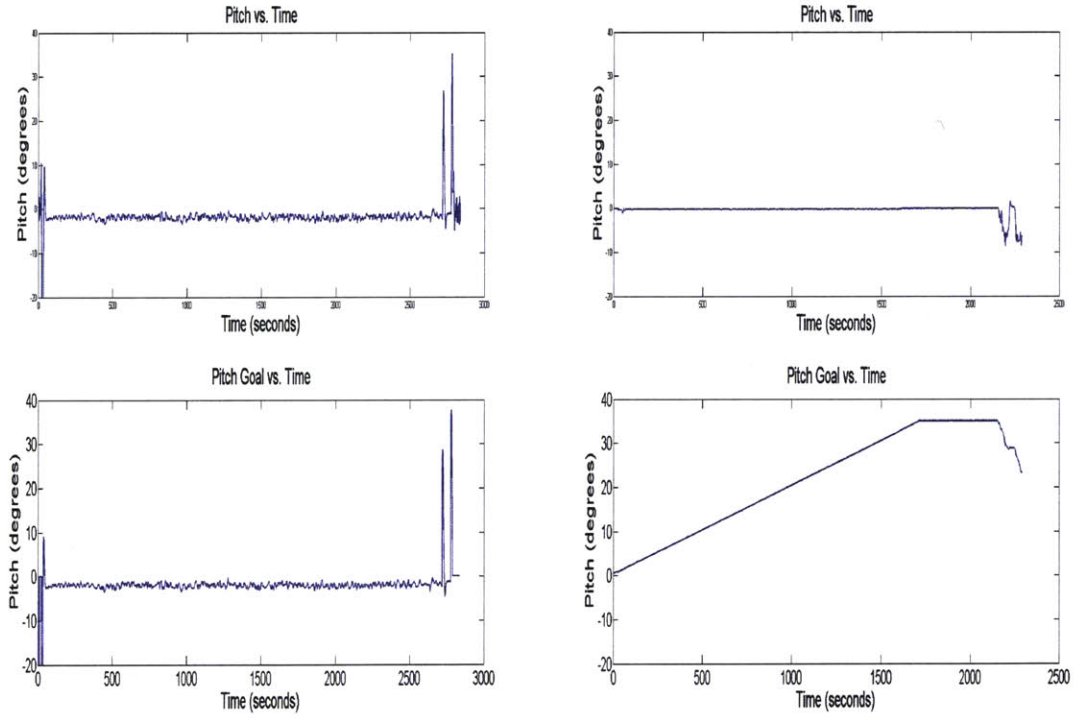
Lastly, the F-measure is the harmonic mean of precision and recall. This is a commonly used metric in information retrieval, and it gives us a single statistic that encapsulates both precision and recall. The harmonic mean is used instead of the arithmetic mean since the harmonic mean better accounts for imbalanced data. For example, consider the case where we have only one mission out of 100 that we label as a fault. If we (unwisely) select a classifier that classifies all of our data instances as anomalous, then our recall is $= 1$ and our precision is $\frac{1}{100}$. The arithmetic mean gives of these two measures gives us $\frac{1+0.01}{2} \approx 0.5$. The F-measure, or harmonic mean, of this strategy gives $\frac{2*1*0.01}{0.01+1} \approx 0.02$, which is a much better evaluation of our classifier[18]. We will use the F-measure as the primary evaluation of our anomaly detection algorithms.

6.2 REMUS Pitch Data Analysis

Our first experiment is identifying faulty pitch data in the REMUS 600 data set. The first step is to generate labels for each mission in our test set, which will allow

us to assess performance. In order to classify missions as normal or anomalous, we analyzed time series plots of pitch output and pitch goal. We classify a mission as a fault if the pitch output appears to deviate significantly from the pitch goal.

Figure 6-1 shows examples of pitch data from two missions. Figure (a) illustrates a mission in which the pitch remained in sync with the pitch goal for the entirety of the mission. Figure (b) shows a mission in which the pitch goal continually increases, while the pitch output remains constant. The mission from figure (a) was classified as normal, while the mission corresponding to figure (b) was classified as anomalous. After consulting the fault log from mission (b), we noted that the vehicle was stuck on the surface, and unable to submerge for the entirety of the mission. Since the anomalous data is explained in the fault log, we classify mission (b) as a modeled fault.



(a) (b)

Figure 6-1: Example of (a) Normal Pitch Data and (b) Faulty Pitch Data

After analyzing time series plots for each mission, we found that the REMUS 600 data contained 31 faults out of a total of 267 missions. Seven of these faults were not explained by the corresponding mission fault log. These missions were classified as unmodeled faults.

Experimental Procedure

The steps that were taken in this experiment are as follows:

1. Label each mission as anomalous or normal, as described above.

2. Compute features used for anomaly detection for each mission. Let $P_i(t)$ be the pitch output for mission i at time t. Let $G_i(t)$ be the pitch goal for mission i at time t. For this experiment we used the following two features:

$$\begin{aligned} x_i^{(1)} &= \text{mean}(P_i(t) - G_i(t)) \\ &= \frac{\sum_{t=1}^{T} P_i(t) - G_i(t)}{T} \\ x_i^{(2)} &= \text{standard deviation}(P_i(t) - G_i(t)) \\ &= \sqrt{\frac{\sum_{t=1}^{T}((P_i(t) - G_i(t)) - x_i^{(1)})^2}{T}} \end{aligned}$$

3. Normalize data. Note that this is an optional step when implementing these techniques. We choose to normalize our data in this experiment so that our features are on the same scale.

4. Divide the data into a training set, $D_{training}$ and D_{test}. We used $\frac{2}{3}$ of our data set for training.

5. Remove known faults from $D_{training}$, since we will not use missions with known faults to generate our models for normal data. In practice, we will be able to

remove modeled faults because they appear in the fault log. Note that $D_{training}$ may still contain unmodeled faults. We will keep modeled faults in D_{test} in order to gauge classifier performance on all types of faults.

6. Use our trained model and corresponding anomaly detection algorithm to generate classifier, $\hat{f}(x)$.

7. Classify data instances in D_{test} using $\hat{y}_i = \hat{f}(x_i)$.

8. Compute performance metrics by comparing $\hat{y}_i$ and y_i for each mission i in our test set.

6.2.1 Model Selection

KDE Model

In chapter 5, we discussed heuristics for selecting the bandwidth parameter, h, for kernel density estimation. We now look at how varying our threshold, α, affects our performance metrics over the test set.

Figure 6-2 shows our KDE model performance on our training set. This was obtained by finding decision boundaries corresponding to various confidence intervals between 99.9% ($\alpha = 0.001$) and 85% ($\alpha = 0.15$). We achieve the highest F-measure for $\alpha = 0.025$ and $\alpha = 0.075$.

One important observation is that our results are relatively constant (i.e. F-measure ≈ 0.6) for α values between 0.01 and 0.1. This tells us that we can still achieve near optimal results for a range of α values. In practice, α must be selected without having access to data labels, so having a range of α values that achieve good results is crucial.

LOF Model

For our LOF experiment, we fit models of various k values and $Threshold_{LOF}$ values. Recall that $Threshold_{LOF} = n$ implies that our anomaly output threshold corresponds to the n^{th} highest anomaly score of data instances in our test set. We

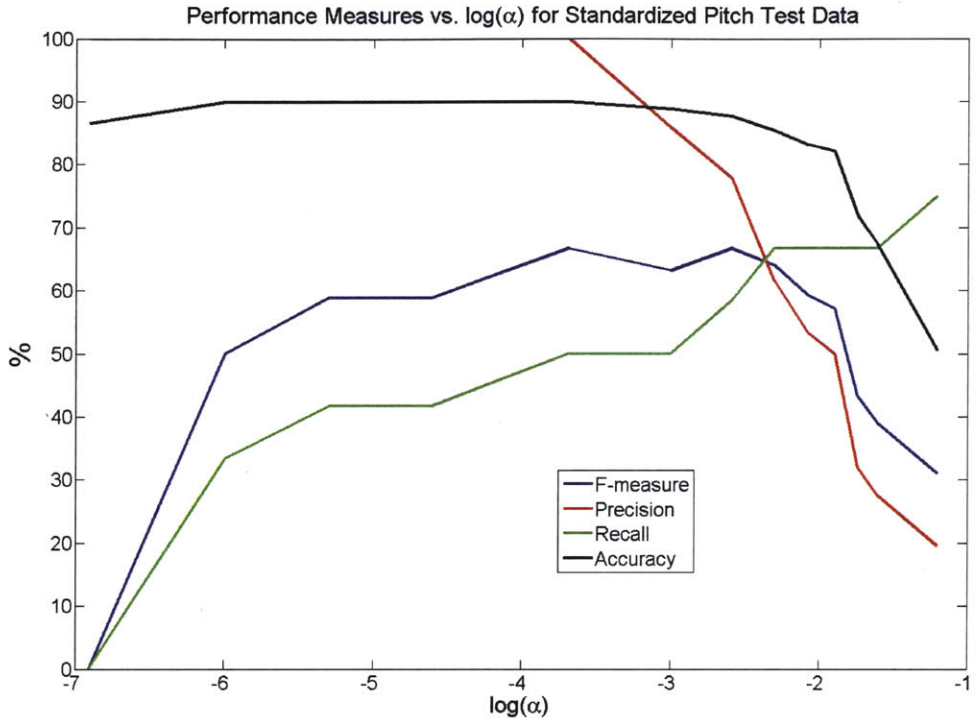


Figure 6-2: Performance Measures vs. $log(\alpha)$ for REMUS 600 Pitch Test Data Features

performed a grid search to find the optimal k and $Threshold_{LOF}$ based on F-measure. We find that we achieve the highest F-measure for $k = 28$ and $Threshold_{LOF} \in \{2, 3\}$. Figure 6-3 shows our performance measures for $k = 28$ for thresholds between 0 and 10.

One observation from figure 6-3 is that our results drop dramatically for slight changes in $Threshold_{LOF}$. Changing our threshold from 2 to 3, our F-Measure drops from 74% to around 40%. Again, in practice, we must select our parameters prior to classifying test instances. Thus, unless we have a systematic and intelligent way to determine our parameters beforehand, we will likely not achieve results that are close to optimal. Figure 6-4 shows F-measure performance for constant $Threshold_{LOF}$ as a function of k. Again, we notice a lot of variation in our results.

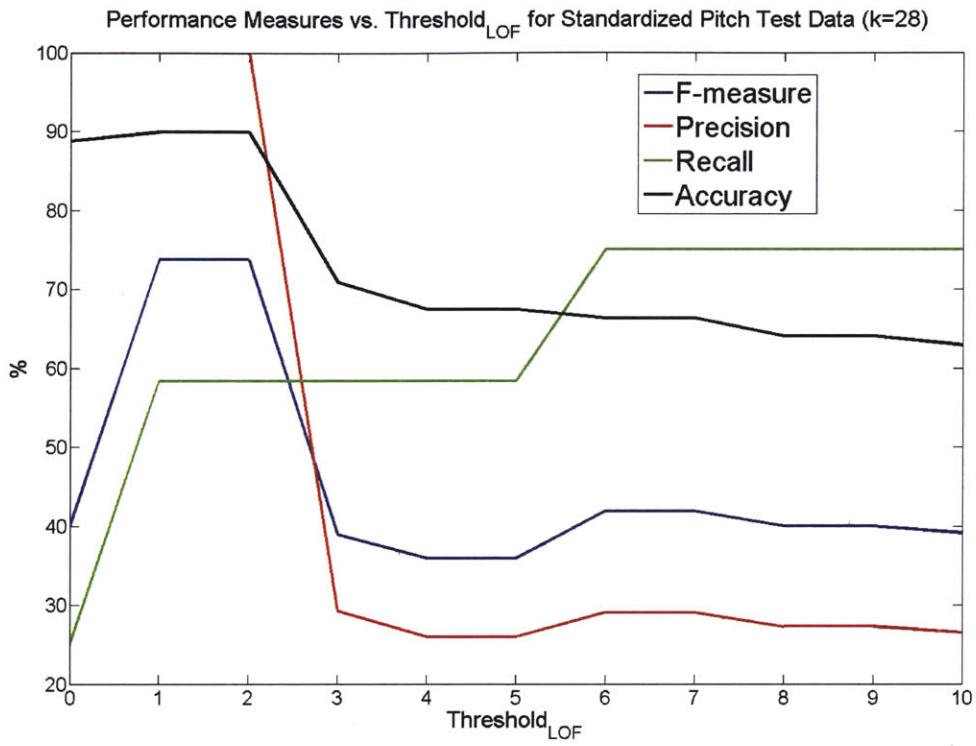


Figure 6-3: Performance Measures on Pitch Test Data vs. $Threshold_{LOF}$ for optimal k.

6.2.2 Results

We present results for selected KDE and LOF models. Table 6.2 shows performance metrics for KDE models with $\alpha = 0.025$ and $\alpha = 0.075$ and for LOF model with $k = 28$ and $Threshold_{LOF} = 2$. We find that the best LOF model outperforms our KDE models in nearly every category. However, recall that we only fit KDE models with constant bandwidth parameters, whereas we performed a grid search over all LOF parameters.

Also, as previously noted, the performance of our LOF models varied significantly with slight adjustments to our parameters. This would indicate that achieving good performance with LOF would be much more difficult, since parameters would have to specified without any information about performance over a labeled test set.

Another advantage of the KDE models is that we can easily illustrate decision boundaries for bivariate data. Figure 6-5 shows our decision boundaries for $\alpha = 0.025$

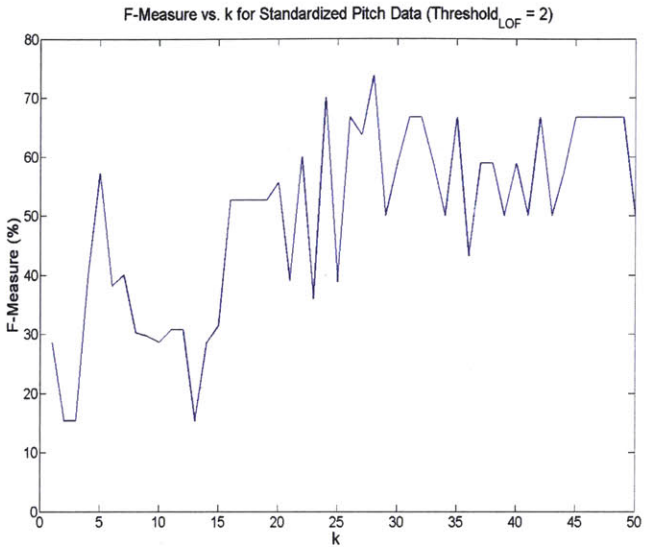


Figure 6-4: F-Measure vs. k for $Threshold_{LOF} = 2$

	KDE ($\alpha = 0.025$)	KDE ($\alpha = 0.075$)	LOF (Best Performance)
F-measure	0.67	0.67	0.74
Recall	0.50	0.58	0.58
Precision	1.0	0.78	1.0
Accuracy	0.90	0.88	0.90

Table 6.2: KDE & LOF Performance Measures for REMUS Pitch Data Experiment

and $\alpha = 0.075$ with both training and test sets. As expected, our "normal" region is slightly larger for $\alpha = 0.025$.

6.3 REMUS Thruster Data

We now attempt to identify faults in REMUS thruster data. Again, our first step is to label each mission as faulty or normal by analyzing time series data from each mission. Figure 6-6 illustrates both normal time series data (a) and time series data representing a faulty mission (b). For mission (a), the thruster rpm matches the thruster goal for the entirety of the mission. For mission (b), however, the thruster rpm goal reaches 1000, but the thruster rpm output cannot match this goal, and remains around 800 rpms. This fault is not listed in the fault log, and we thus classify the mission as an unmodeled fault. It is likely that the thruster rpm goal was

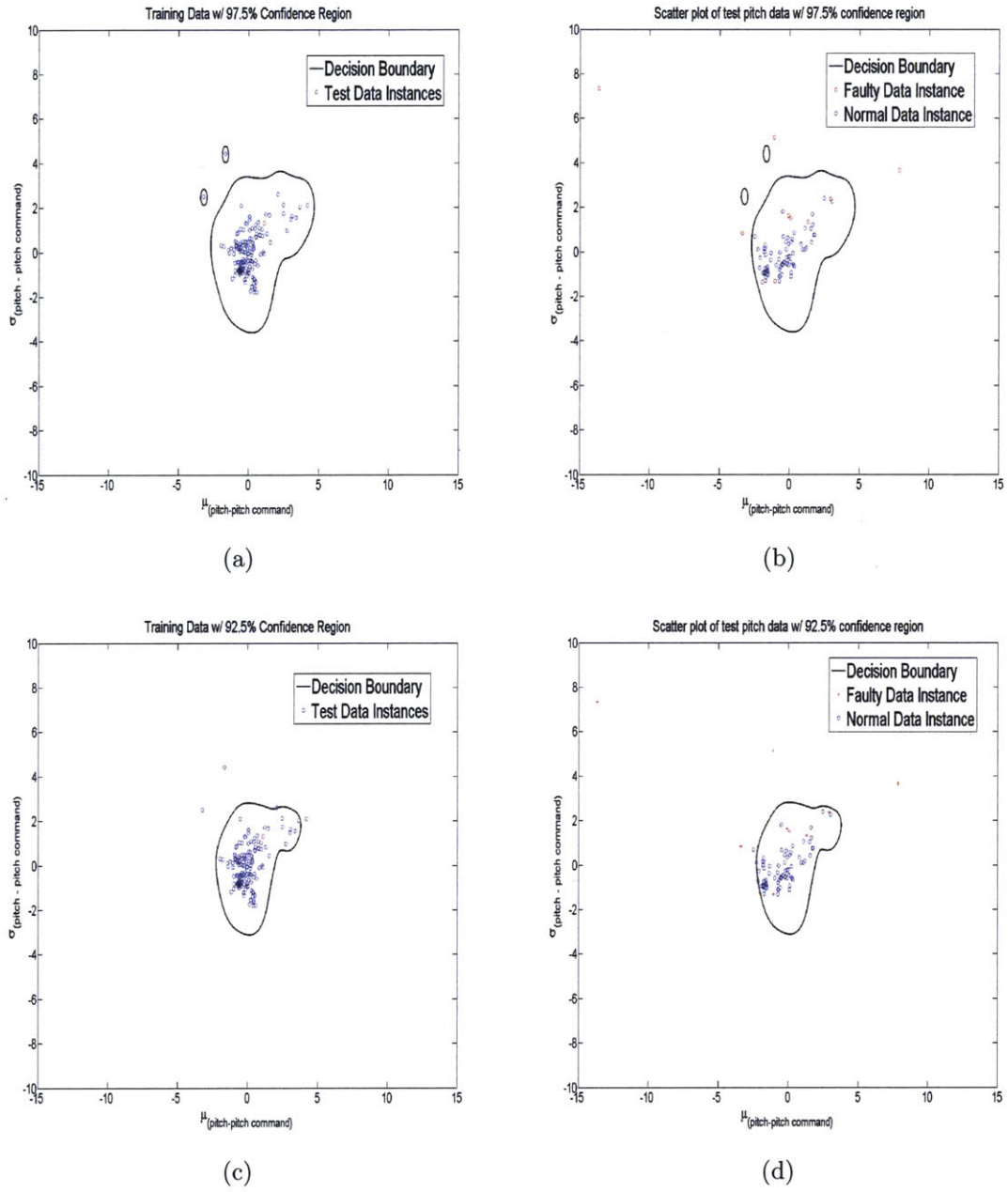
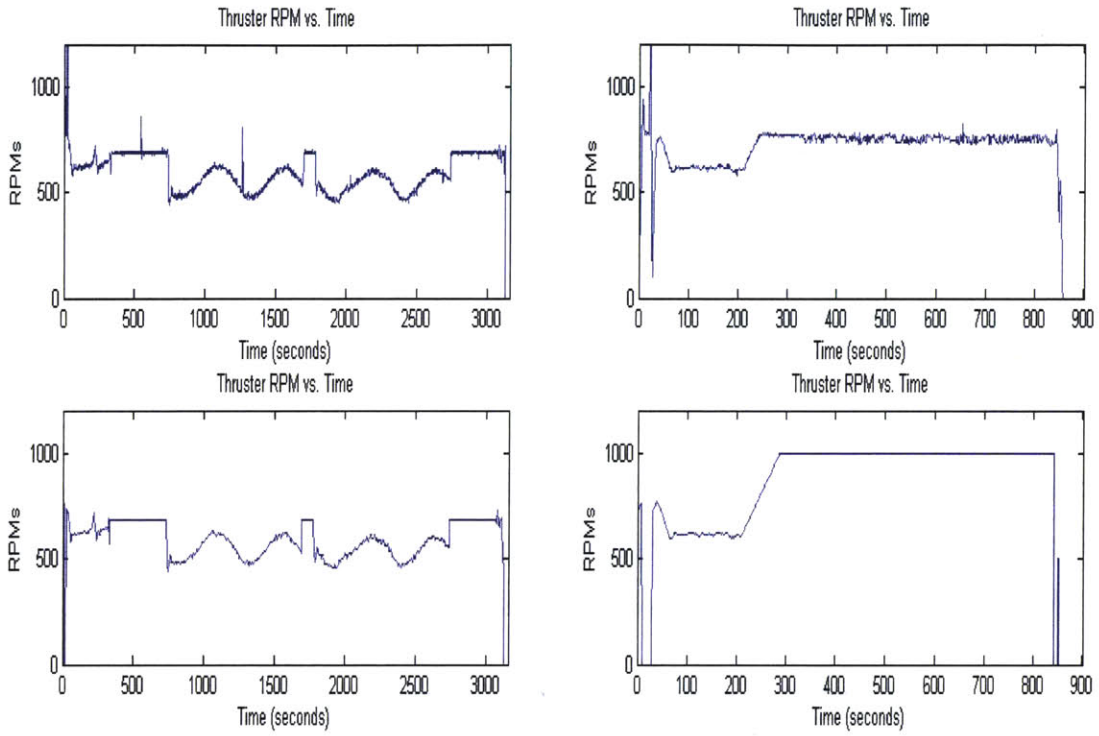


Figure 6-5: Pitch Data Decision Boundaries with (a) Training Data and $\alpha = 0.025$ (b) Test Data and $\alpha = 0.025$ (c) Training Data and $\alpha = 0.075$ (d) Test Data and $\alpha = 0.075$

erroneously set too high for several legs of the mission.

Our entire data set contains 32 faults, 16 modeled, and 16 unmodeled. Our test set contains 5 unmodeled faults, which remain in the set when we generate our KDE and LOF models.



(a) (b)

Figure 6-6: Example of (a) Normal Thruster Data and (b) Faulty Thruster Data

Experimental Procedure

For this anomaly detection experiment, we follow the same procedure as with the pitch data experiment. In this case, we use the following two features:

$$x_i^{(1)} = \text{mean}(R_i(t) - G_i(t)) \tag{6.1}$$

$$= \frac{\sum_{t=1}^{T} R_i(t) - G_i(t)}{T} \tag{6.2}$$

$$x_i^{(2)} = \text{standard deviation}(R_i(t) - G_i(t)) \tag{6.3}$$

$$= \sqrt{\frac{\sum_{t=1}^{T}((R_i(t) - G_i(t)) - x_i^{(1)})^2}{T}} \tag{6.4}$$

where $R_i(t)$ is the thruster RPM output at time t for mission i, and $G_i(t)$ is the thruster output goal at time t for mission i.

We again use the first $\frac{2}{3}$ of missions for training data, and the remaining missions for testing. Our goal is to assess performance of our models, and analyze how parameter selection affects this performance.

6.3.1 Model Selection

KDE Model

For KDE anomaly detection, we vary α to assess performance for various anomaly detection thresholds. Figure 6-7 shows how our performance metrics vary with respect to α. We find that we achieve optimal F-measure for $\alpha = 0.075$.

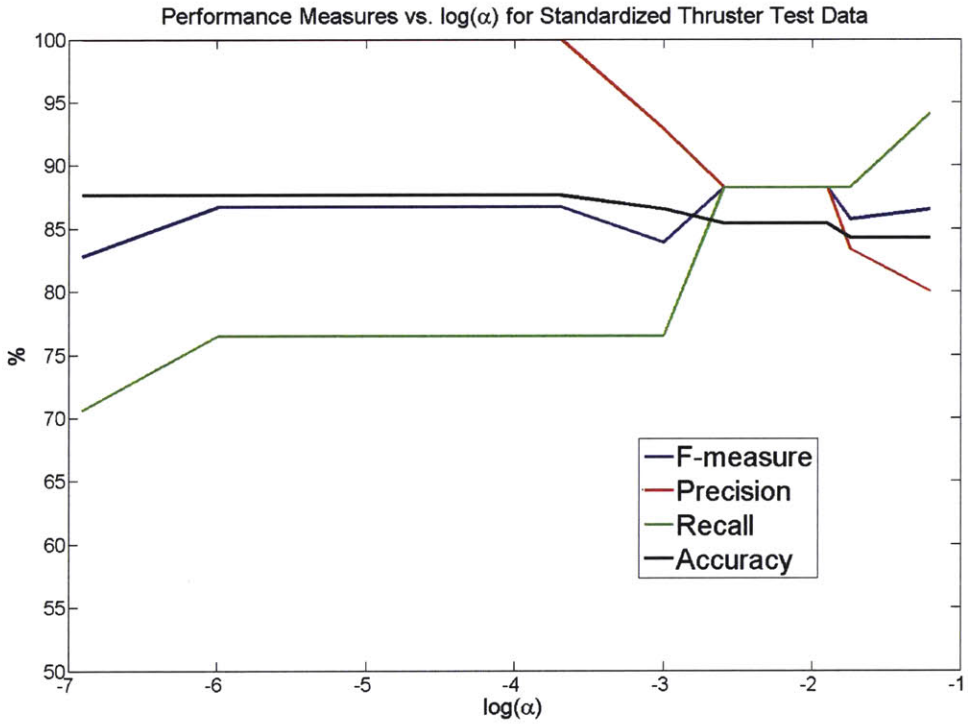


Figure 6-7: Performance Measures vs. $log(\alpha)$ for REMUS 600 Pitch Test Data Features

As with the pitch data experiment, we find that we achieve near optimal results for a relatively wide range of α values. In this case, we achieve an F-measure of

greater than 0.85 for α between 0.025 and 0.3.

LOF Model

For our LOF technique, we assess performance for various values of $threshold_{LOF}$ and k. We perform a grid search to find the model that achieves the best performance. For our thruster data, we find that we achieve the highest F-measure for $k = 20$ and $threshold_{LOF} = 4$.

Figure 6-8 illustrates our performance measures for optimal k and values of $threshold_{LOF}$ between 0 and 10. In this case, we find that performance remains near optimal for a wide range of threshold values. However, like with the pitch data experiment, varying k slightly leads in significantly worse performance. Again, this is problematic, since, when applying this technique in the real world, we must decide on parameters for our LOF model without knowledge of data labels or our test set.

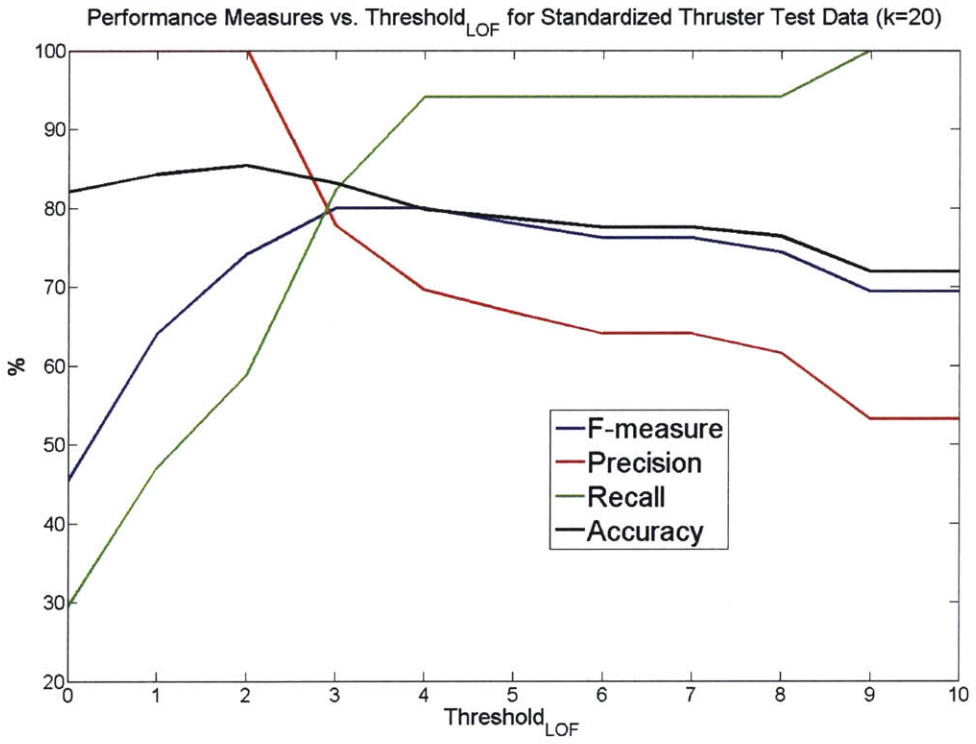


Figure 6-8: Performance Measures on Thruster Test Data vs. $Threshold_{LOF}$ for optimal k.

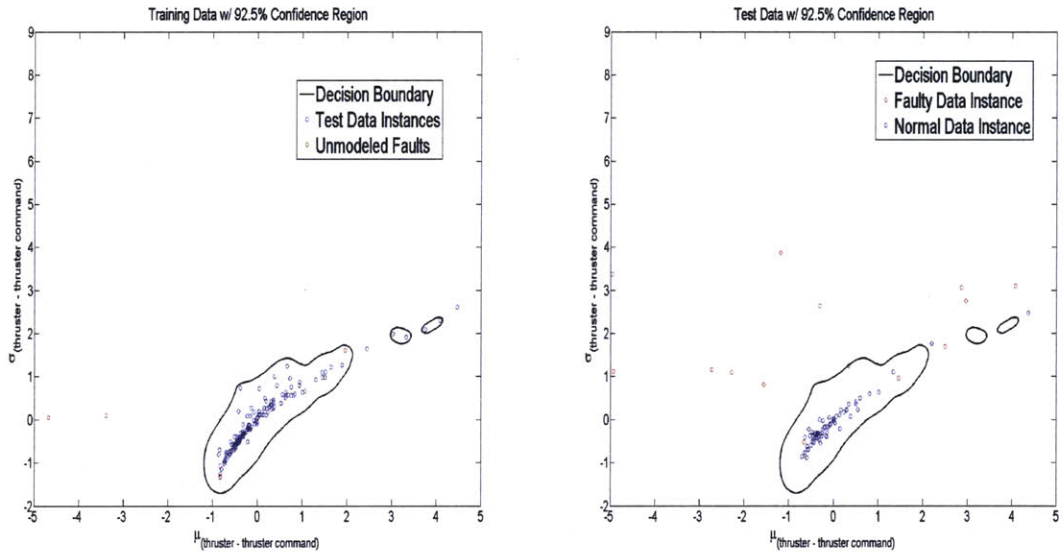
6.3.2 Results

We now present results for selected KDE and LOF models. Figure 6.3 shows performance metrics for models that achieved the highest F-measure. For this experiment, we find that our KDE model outperforms the LOF model.

	KDE ($\alpha = 0.075$)	LOF (Best Performance)
F-measure	0.88	0.74
Recall	0.88	0.58
Precision	0.88	1.0
Accuracy	0.85	0.90

Table 6.3: KDE & LOF Performance Measures for REMUS Pitch Data Experiment

Figure 6-9 illustrates the decision boundaries corresponding to $\alpha = 0.075$. This corresponds to the best performing KDE anomaly detection model.



(a) (b)

Figure 6-9: Thruster Data Decision Boundaries with (a) Training Data and $\alpha = 0.075$ (b) Test Data and $\alpha = 0.075$

6.4 MARV Thruster Data

Our last anomaly detection experiment is on MARV thruster data. As discussed in Chapter 2, the MARV data set contains performance data from 128 missions performed by NUWC-NPT operators in 2013. After analyzing time series data from each mission, we classified 27 out of 128 missions as faults. The analysis on the time series data was done in a similar manner as in section 6.3 (i.e. by comparing thruster rpm output and thruster rpm goal).

Experimental Procedure

The features used in this experiment are identical to those used in the REMUS thruster data experiment. Our first feature is the arithmetic mean of difference between the thruster output and thruster goal, averaged over the entire mission (equation 6.2). The second feature is the standard deviation of the difference of the thruster output and thruster goal (equation 6.4).

One difficulty when performing this experiment was differentiating between modeled faults and unmodeled faults. In section 6.3, we described the MARV fault log, and explained the difficulty in pinpointing events that cause abnormal performance data. Because of this difficulty, we were not able to classify faults as modeled or unmodeled. For this experiment, we left all anomalies in the test data and training data. We used $\frac{2}{3}$ of the mission data for training, and the rest for assessing model performance.

6.4.1 Model Selection

KDE Model

Again, we assess performance of our KDE model by computing performance metrics over our test set. Figure 6.4 illustrates our performance metrics for various α values. Optimal performance (F-measure = 0.88) is obtained for $0.05 \leq \alpha \leq 0.2$. As shown in Figure 6-10, optimal or near optimal performance for a wide range of α values.

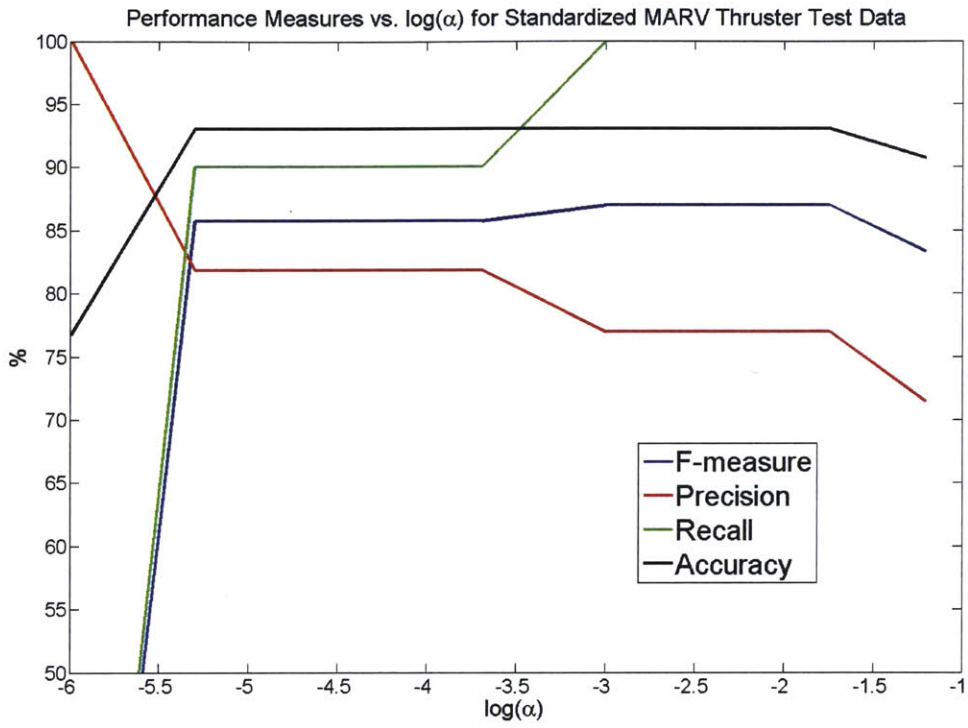


Figure 6-10: Performance Measures vs. $log(\alpha)$ for MARV Pitch Test Data Features

LOF Model

We again assess the performance of our LOF anomaly detection technique for various values of $threshold_{LOF}$ and k. By performing a grid search for integer values $k \in \{1, 2, \ldots, 30\}$ and $threshold_{LOF} \in \{1, 2, \ldots, 10\}$, we find that optimal performance (F-Measure $= 0.86$) is achieved for $k = 13$ and $threshold_{LOF} = 7$.

As in previous experiments, we found that slight changes in our parameters can lead to large deviations in classification performance. We omit plots for performance measures since they are nearly identical to figures 6-4 and 6-3.

6.4.2 Results

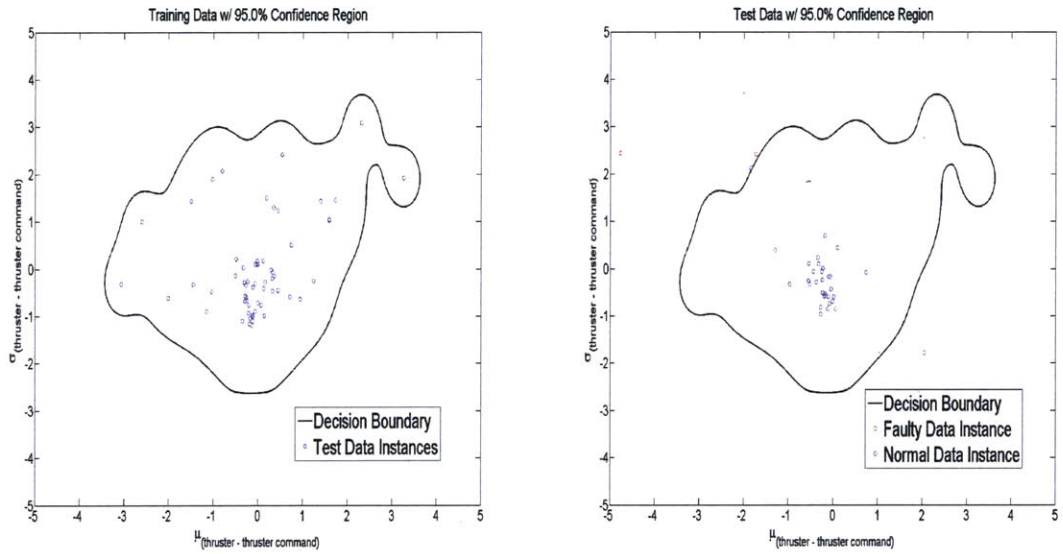
We now present performance results for selected KDE and LOF models over our MARV thruster data test set. Figure 6.4 shows that we achieve slightly better results for our KDE model.

We now illustrate our decision boundaries for our optimal KDE model. Figure

	KDE ($\alpha = 0.05$)	LOF (Best Performance)
F-measure	0.87	0.86
Recall	1.00	0.90
Precision	0.77	0.82
Accuracy	0.93	0.93

Table 6.4: KDE & LOF Performance Measures for MARV Pitch Data Experiment

6-11 shows our decision boundaries with both training set and test set for $\alpha = 0.05$.



(a) (b)

Figure 6-11: MARV Thruster Data Decision Boundaries with (a) Training Data and $\alpha = 0.05$ (b) Test Data and $\alpha = 0.05$

6.5 Discussion

In this chapter, we implemented anomaly detection using KDE and LOF techniques. In order to assess the performance of our models, we generated labels for selected UUV systems (i.e. thrusters and pitch control) for the REMUS 600 and MARV data sets. The goal of these experiments was to compare the performance of these techniques using real data sets, and to analyze how parameter selection affects performance.

For the REMUS pitch data experiment, the optimal LOF model slightly outperforms all of our KDE models. It is important to note, however, that for our KDE

models, we only varied our detection threshold, and we used a constant bandwidth parameter based on the heuristic discussed in Chapter 5. For our LOF implementations, we varied both the detection threshold and the k parameter. It is possible that better results could be achieved for our KDE models by adjusting our bandwidth parameter. For both the MARV and REMUS 600 thruster data experiments, we found that the KDE outperformed LOF.

Perhaps the most important takeaway from these experiments is that our KDE models achieved near optimal performance for a wide range of α values. For LOF, on the other hand, we found that results vary greatly with slight adjustments to our parameters. This indicates that KDE would be more applicable in practice, since parameters must be selected prior to classifying test data instances.

Lastly, we can also use the results of these experiments to make a recommendation on α values to use in future implementations. In all experiments we achieved optimal or near optimal results for α between 0.025 and 0.1.

Chapter 7

Extending KDE Anomaly Detection Technique

Thus far, we have focused on the problem of identifying anomalous performance data for a particular UUV mission. Our KDE anomaly detection technique seems to be a reasonable approach to building a classifier that can automatically identify such anomalies. In this chapter, we explore ways in which we can extend our KDE anomaly detection approach to serve other purposes. In particular, we attempt to answer the following questions:

1. **Incorporating New Data:** We would like to update our KDE model as we receive more performance data from UUV missions. One issue with incorporating new data is that UUV systems may experience degradation over time. This degredation will likely appear as an anomalous trend in certain features of our data. If we update our KDE model after each mission, we may not identify such anomalous trends. How can we smartly incorporate new vehicle data while accounting for potential anomalous trends?

2. **Discrete Data:** We have utilized KDE for estimating the data generating process for continuous data. Can we use similar anomaly detection methods for discrete valued data?

3. **Reusing KDE Models for Other Vehicles:** Suppose that we have a new

UUV (or newly reconfigured UUV) without sufficient historical data to train a KDE model. Can we use KDE models from other UUVs to identify anomalies in these vehicles?

7.1 Incorporating New Vehicle Data

In this section, we discuss how we can smartly incorporate new data into our KDE models. We can imagine a scenario in which we have a limited amount of historical data to train our KDE models. We would like to include more data into our KDE model as UUVs complete more missions in order to improve upon our PDF approximation.

Anomalous Trends

According to experienced UUV operators, there have been issues in the past with UUV systems experiencing performance degradation over time. This may lead to anomalous trends in our data features. When looking at data on a mission by mission basis, these trends may not be readily apparent. However, when looking back over the course of a number of missions, it becomes clear that a system is not behaving as expected.

Normalization of Deviance

Furthermore, there have been occasions where operators become accustomed to systems that are performing sub-optimally. One case study that we explored in this research is the Columbia Space Shuttle Disaster of 2003. The Space Shuttle Columbia disintegrated while reentering the Earth's atmosphere, and it was later discovered that the cause of the accident was due to damage to the shuttle's heat shield that occurred during launch. Specifically, pieces the thermal insulation foam that covered the external tank broke off during launch. A large piece of the insulation struck the shuttle's left wing, causing damage to reinforced carbon-carbon panels[3]. This damage allowed hot gasses to enter the shuttle during reentry, causing the vehicle to

disintegrate.

Although design requirements stated that foam shedding should not occur during launch, several previous Columbia missions experienced a similar amount of foam shedding. These previous missions, however, were deemed successful, as foam shedding did not lead to significant damage. Over time, NASA engineers and management became accustomed to foam shedding events, and dismissed them as "accepted risk"[3]. This phenomenon of accepting events that should not occur was termed "normalization of deviance" by sociologist Diane Vaughan.

The case of the space shuttle Columbia disaster, we believe, is relevant in the problem of identifying anomalous trends in UUV mission performance data. The foam shedding incidents that occurred can be viewed as "anomalies". Similar to Columbia engineers, UUV operators can become accustomed to these anomalies that occur when a system is not behaving as originally expected.

In order to ensure that anomalous trends do not become accepted as normal by our KDE anomaly detection algorithm, we must be careful when incorporating new performance data into our KDE approximations. In a scenario in which we have a limited amount of data and only a few vehicles from which to build our KDE models, an anomalous trend that is present in one or more vehicles may not be correctly identified. We now look at an example using fabricated data containing an anomalous trend, and we discuss ways in which we can identify anomalous trends in test data before we incorporate the data into our models.

Anomalous Trend Example

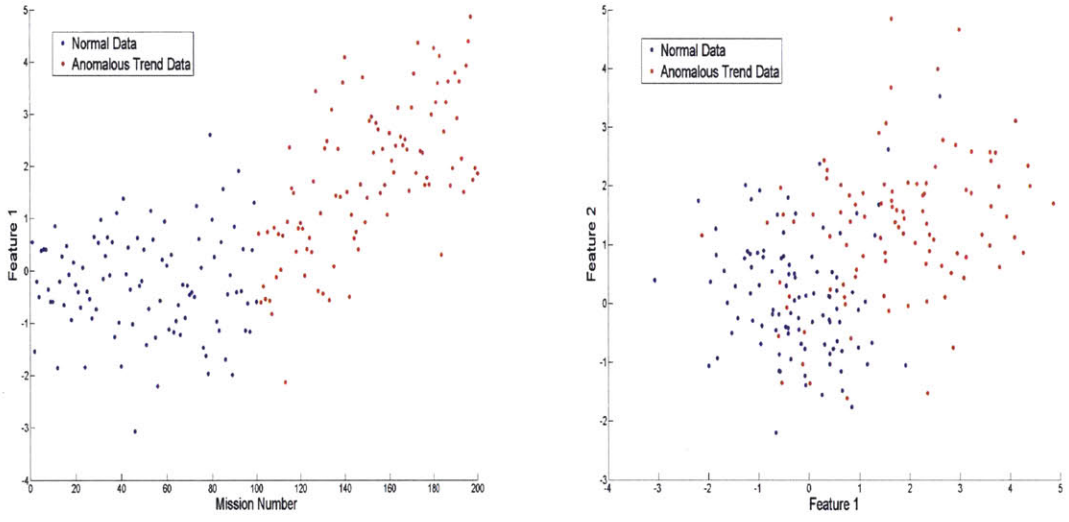
In this example, we use fabricated data to show how an anomalous trend might cause our KDE model to incorrectly classify anomalous data instances. In this scenario, suppose we have performance data from one vehicle, and we are using a bivariate feature space to perform anomaly detection. We have used the first 50 missions, consisting of normal operational data, to build our KDE model. An anomalous trend begins at mission 100. Our normal data was generated using a bivariate Gaussian distribution. Our anomalous data is generated from the same bivariate Guassian plus

some function of mission number. That is, for normal data points, x_n, and anomalous data points, x_a, we have

$$x_n \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$
$$x_a \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) + g(Mission\ Number),$$

where $g(Mission\ Number)$ increases linearly with the mission number.

Figure 7-2 (a) shows *feature 1* as a function of mission number. The first 100 missions (blue) are normal, and the last 100 missions (red) exhibit an anomalous trend. Figure 7-1 (b) shows are bivariate feature space, again with normal data in blue, and anomalous trend data in red.



(a) (b)

Figure 7-1: Illustration of (a) Anomalous Trend in Feature 1 and (b) Bivariate Data Set in Anomalous Trend Example

Since we have used relatively few missions (50) to generate our KDE model, we would like to add more data as we receive it. In practice, if we classify a mission as anomalous, an operator would check time-series plots to see if a fault has occurred. However, if we are frequently updating our KDE model, our PDF estimate may

increase in the direction of our anomalous trend before we correctly identify the trend. This is similar to the idea of "normalization of deviance", as our classifier will learn to accept anomalous data as normal.

Figure 7-2 demonstrates this idea using our fabricated data. The figure shows how our decision boundary gradually expands in the direction of the anomalous trend. The initial decision boundary is based only on the normal data, while the larger decision boundaries are based on normal data in addition to a subset of anomalous data.

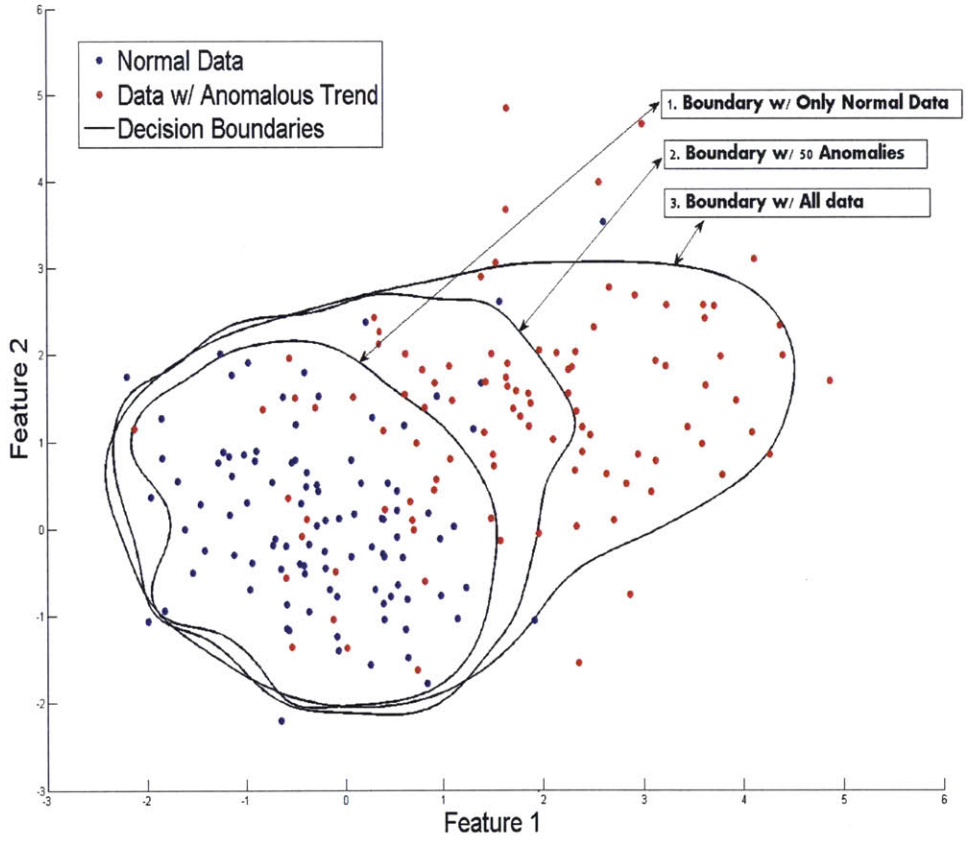


Figure 7-2: Decision Boundaries for Various Training Sets in Anomalous Trend Example

Strategies for Accounting for Anomalous Trends

We now discuss strategies for incorporating new data into our models while accounting for potential anomalous trends in the data.

1. **Do not incoporate new data into our original KDE model:** If we do not update our original KDE model, then anomalous trends such as the one in the example above will eventually be correctly classified as anomalous. Again, this is not ideal, since we would like to improve upon our KDE model as we receive new data. However, if we are confident that our original decision boundaries provide an accurate representation of the normal region, then adding new data may be unnecessary.

2. **Perform KDE in a feature space in which anomalous trends appear as outliers:** One promising strategy would be to use alternative features that account for trends in our data. For example, we could use an average of our features of the course of, say, 5 or 10 missions. If we choose appropriate features, then anomalous trends may appear as point anomalies in our new feature space.

3. **Use a sliding window technique to track frequency of anomalies:** The last strategy that we explored in this research is using a sliding window technique to track the number of anomalies over a certain period. Window based anomaly detection techniques for sequential data are discussed by Chandola in [8]. The idea is to extract fixed length "sliding" windows from a test sequence and assign an anomaly score to each window.

   We now illustrate how a sliding window technique could be implemented for our anomalous trend example. We use a window length of 20, and index our windows by k. The k^{th} sliding window contains missions k through $k+19$. We assign an anomaly score to the k^{th} window as $Anomaly\ Score(k) = number\ of\ anomalies\ identified\ in\ missions\ k\ through\ k+19$. We can identify an anomalous trend by setting a threshold on $Anomaly\ Score(k)$. One way to set a threshold on our anomaly score would be to pick the highest anomaly score for windows

in our test set. The idea is that if a data point never falls into a window with a high anomaly score, then we can be relatively sure that the point is not apart of anomalous trend.

Figure 7-3 shows the number of anomalies vs. our sliding window index for a window length of 20. In this experiment, we added new data to our KDE model after each sequence of 20 missions. As shown in the figure, our anomaly scores typically grow as k increases, up until the point in which we add new data to our KDE model. In practice, we would set a threshold on the anomaly score of our sliding windows and classify a window as anomalous if the number of anomalies in that window is above a certain number. In this example, if we set a threshold of 5 on our anomaly score, then we would correctly identify the anomalous trend by mission 125.

This method also provides an intuitive way to add new data to our KDE model. If data point x_k is never contained in a window that is classified as anomalous, then we can be relatively sure that the data point is not apart of an anomalous trend. In which case, we would add data point x_k to our KDE model. In other words, if our sliding window has passed point x_k and no anomalous window has been identified, then we can add point x_k (and all points x_j for $j < k$) to our KDE model.

7.2 Discrete Data

Thus far, we have focused on identifying point anomalies in continuous data. We now briefly discuss how we can use a similar approach to identifying anomalies in discrete data.

Examples of discrete UUV data in which we are interested in identifying anomalies include:

- The number of times a particular modeled fault occurs during a mission, as contained in the fault log.

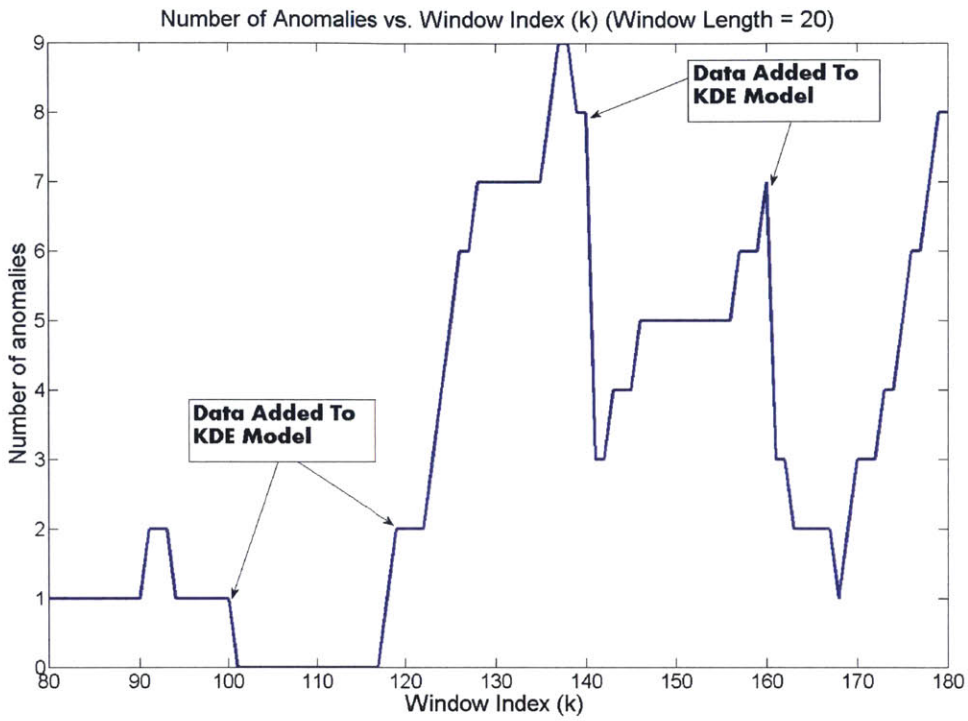


Figure 7-3: Number of Anomalies per Window vs. Sliding Window index for Anomalous Trend Example

- The number of days between unscheduled maintenance for a particular UUV system.

- The number of mission failures that occur during a sequence of missions.

Note that for these examples, point anomalies can be identified through simple limit checking. For example, suppose we are interested in the number of times a particular modeled fault occurs during a mission. We would only identify a mission as anomalous if this number exceeds a certain value. Furthermore, all of these examples involve only univariate data. Thus, the problem of finding anomalies in discrete data is equivalent to selecting a threshold on the value of a discrete univariate data point. This is distinctly different than our KDE anomaly detection approach, where our goal is to identify regions of normality in our feature space.

Our goal is to compute an upper (or lower) limit on values for discrete data of interest, and classify points as anomalous if they exceed (or fall below) this threshold.

For the frequency of modeled faults during a mission and the number of mission failures that occur during a sequence of missions, we would be interested in finding an upper limit. For the number of days before schedule maintenance, we are interested in computing a lower limit.

The simplest way to choose an upper limit would be to select the value that is in the $100 * \alpha$ percentile of our test data, where α is a prior belief in the likelihood of an anomaly. Figure 7-4 illustrates a histogram of "Skipping Motor Cycle" faults from REMUS 6000 missions. These frequencies were obtained by parsing REMUS fault logs for each mission in the REMUS 6000 test set. For this example, α was chosen to 0.1. The threshold was obtained by finding the mission in the 10^{th} percentile of frequency of faults. Since there are 72 missions in our test set, the upper limit was based on the mission with the 7^{th} highest number of faults.

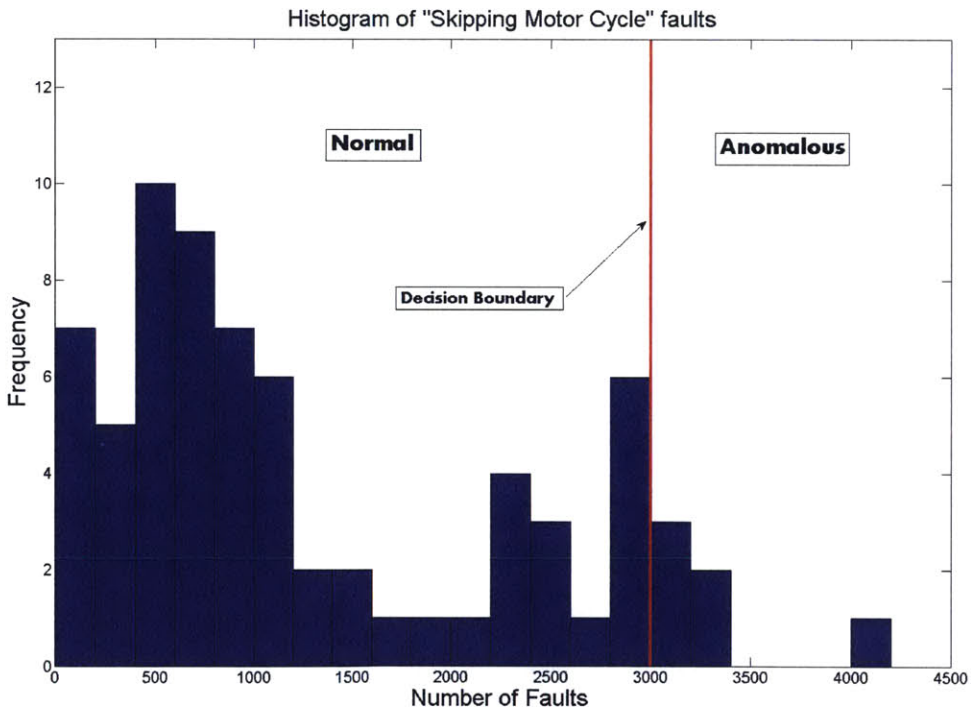


Figure 7-4: Histogram of REMUS 6000 Faults with Decision Boundary

7.3 Reusing Models for New Vehicles

The last question we attempt to answer in this chapter is whether or not we can use previously developed KDE models to perform anomaly detection on a different UUV. This would be of interest in a scenario in which we have a new or recently reconfigured vehicle, and we do not have enough historical data to build a reliable KDE approximation for a data generating process of interest.

In order to explore this question, we computed features of REMUS 6000 and MARV data, and attempted to classify anomalous data instances based on the REMUS 600 models developed in Chapter 6. In addition to computing these selected features, we also analyzed time-series data from these vehicles to generate labels for each mission.

When performing our experiments with REMUS 6000 and MARV data, it became readily apparent that our features did not lie in the same regions as the features from the REMUS 600. We began with the REMUS 6000 pitch data, computing the same features as in Chapter 6 (i.e. mean(pitch-pitch goal) and standard deviation(pitch-pitch goal)). We attempted to standardize these features by using the mean and standard deviation obtained from the REMUS 600 data set. The features from the REMUS 6000 missions, however, did not fall into the same region of our feature space, and each mission was classified as anomalous.

As a secondary experiment, we attempted to standardize the REMUS 6000 data using the mean and standard deviation of the REMUS 6000 data itself. This would not be practical in a real-world scenario, as we would likely not have enough data to get reliable estimates for the mean and standard deviation of our features. If we did have sufficient data, we would simply use that data to build a KDE model for the new vehicle. Nevertheless, we attempted to use the same classifier from the REMUS 600 pitch data to identify anomalies in REMUS 6000 pitch data.

Figure 7-5 shows our standardized REMUS 6000 pitch data features with the optimal REMUS 600 decision boundary found in Chapter 6.2. As illustrated in the plot, the decision boundary does not provide a good representation of the normal

region of the REMUS 6000 data. We do classify several anomalies correctly, but our normal region is far too large to be deemed a reliable classifier. These experiments were reproduced for MARV thruster data, but results were even less promising.

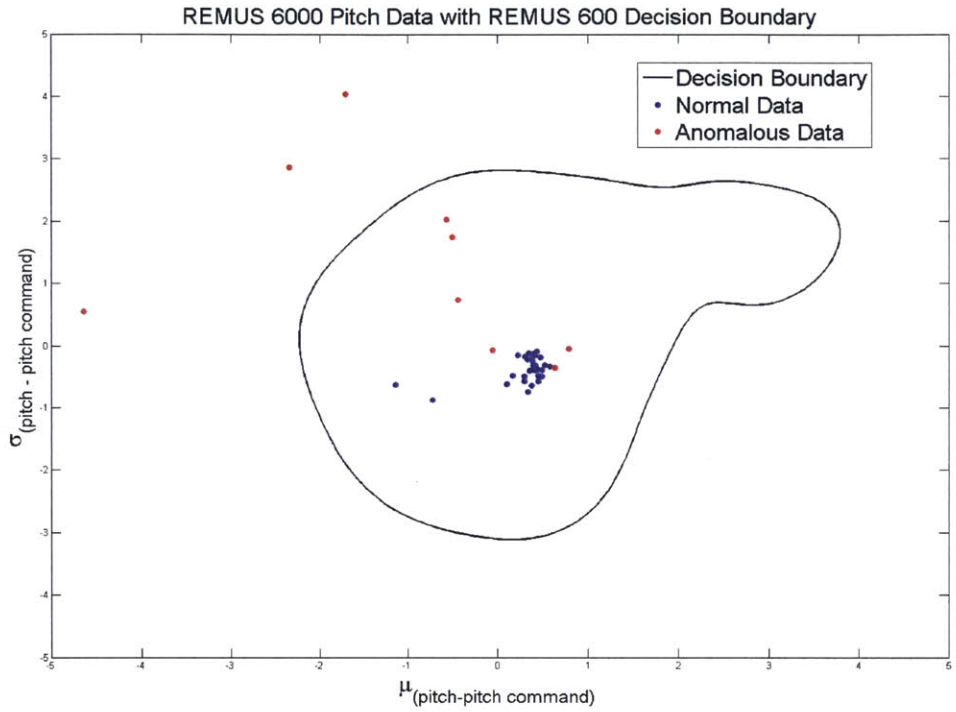


Figure 7-5: REMUS 6000 Pitch Data Features with Previous REMUS 600 Decision Boundary

7.4 Discussion

In this chapter, we discussed ways to extend our KDE anomaly detection approach for various purposes. We first discussed strategies for incoporating new UUV performance data to improve upon our KDE models while accounting for potential anomalous trends in our data. We explored using a sliding window technique for identifying these anomalous trends, and illustrated results using fabricated data.

We also discussed the problem of identifying point anomalies in discrete data. For discrete UUV data, limit checking appears to be a simple, yet reliable way to classify anomalous data.

Lastly, we explored the possibility of using performance data from one vehicle to identify anomalies in another vehicle. We experiments used features from the REMUS 6000 and MARV data sets with decision boundaries computed from REMUS 600 data. The results were not promising, as the data generating processes for the features of interest were significantly different between the vehicles.

Chapter 8

Conclusion

The goal of this thesis is to describe a tool that can be used help UUV operators identify anomalous features of UUV performance data. These anomalies are of interest because they are often the result of a fault, or an abnormal condition that can cause an element of a UUV system to fail. Our approach was to use one-class classification methods to build classifiers based on historical data. We make the assumption that the majority of our training data comes from normal UUV operations.

Our primary approach was to use kernel density estimation to build a probability model for data generating processes of interest. We classify future missions as anomalous if data features from these missions lie in areas of low probability. Our kernel density anomaly detection algorithm was compared with another method, local outlier factor. In order to provide insight on the effectiveness of these algorithms, we provided experimental results for selected UUV systems and data features.

8.1 Summary of Results and Contributions

Chapter 2 of this thesis provides background information on the UUVs and data sets that were used in this research. We briefly describe the operational capabilities and specifications for both the REMUS series of vehicles and the MARV vehicle. We also describe the data sets that were obtained from WHOI and NUWC-NPT.

In chapter 3, we describe the anomaly detection problem in its most general form.

The purpose of this chapter was to discuss how the effectiveness of an anomaly detection algorithm is largely dictated by the specific characteristics of our particular problem and data sets. We provide analysis of the characteristics of the problem of detecting anomalies in UUV performance data, and determine that one-class classification methods are suitable for our research.

In chapter 4, we explore existing one-class classification methods that might be suitable for UUV anomaly detection. In particular, we discuss statistical methods (both parametric and non-parametric), distance based methods (e.g. k - nearest neighbors and clustering), and the one-class support vector machine. We determine that kernel density estimation and local outlier factor are the two most suitable approaches for our problem.

Chapter 5 is a discussion of parameter selection for kernel density estimation anomaly detection. We discuss the Bowman and Azzalini heuristic for computing a bandwidth parameter directly from our data. We also show how to compute decision boundaries based on our KDE approximations.

Chapter 6 provides experimental results for KDE and LOF anomaly detection on selected features of UUV performance data. In order to gauge the effectiveness of these algorithms, we decided to go through time-series data from each mission in our data sets and label each mission as anomalous or normal. We compute performance metrics by comparing the output of our classifiers with the labels that we generated by analyzing time-series plots.

In chapter 7, we discuss how we can extend our KDE anomaly detection approach for other purposes. We explore methods for identifying potential anomalous trends that may arise from the performance degradation of a particular UUV system. We also discuss the problem of identifying anomalies in discrete data.

8.2 Future Work

One of the key characteristics of the UUV anomaly detection problem is that labeled training data for both the normal and anomalous classes is not readily available. For

this reason, we took a one-class classification approach, and made the assumption that anomalies in our training set are rare occurrences. One idea for future research would be to determine if we can improve upon classification performance by using two-class classification methods (e.g. two-class support vector machine, statistical modeling for both normal and anomalous class, etc.). In order to test this, one would need training data from the normal and anomalous class, which would require analyzing a large amount of time-series data to generate labels. Our preliminary hypothesis is that classification performance would not improve significantly through two-class classification. In our research, we observed that anomalies are typically scattered throughout our feature space with high variance. Thus, attempting to model the anomalous class would likely be ineffective.

Another area for future research would be to improve upon the KDE anomaly detection approach given in this thesis. Classification performance could perhaps be improved by using more relevant features, or by using alternative methods for bandwidth selection. Developing better features would likely require a higher level of technical knowledge about the UUV systems of interest. One approach to bandwidth selection that we did not have the time to explore in this research is using a variable bandwidth parameter. With a variable bandwidth, the value of the bandwidth parameter is a function of the location of the kernel function in a feature space. Using a variable bandwidth parameter could be beneficial if there are regions of our feature space in which normal data exhibits less variance.

Lastly, future research could also include experimentation for other one-class classification methods discussed in Chapter 4.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Kongsberg maritime image gallery. http://www.km.kongsberg.com/ks/web/nokbg0238.nsf/AllWeb/0C6E508198ADAA3EC125774C003C1A6F. Accessed: 2015-02-24.

[2] Richard Bashour, Michael Ansay, and Daniel French. NUWC's mid-sized autonomous reconfigurable vehicle (MARV): Sub surface ship launch & recovery of a UUV efforts. https://www.navalengineers.org/SiteCollectionDocuments/2010December 2010. Accessed: 2015-02-24.

[3] Columbia Accident Investigation Board. Report of columbia accident investigation board, 2003.

[4] Adrian W. Bowman and Adelchi Azzalini. *Applied Smoothing Techniques for Data Analysis.* Oxford University Press Inc., 1997.

[5] Markus M. Breunig, Hans-Peter Kriegel, and Raymond T. Ngand JÃűrg Sander. Lof: Identifying density-based local outliers. *Proceedings of 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[6] Robert W. Button, John Kamp, Thomas B. Curtin, and James Dryden. *A Survey of Missions for Unmanned Undersea Vehicles.* RAND Corporation, Santa Monica, CA, 2009.

[7] Naval Undersea Warfare Center. Unmanned undersea vehicles. http://auvac.org/uploads/platform_pdf/NUWC%20UUV%20-%20Vehicles_2.pdf. Accessed: 2015-02-24.

[8] Varun Chandola. *Anomaly Detection for Symbolic Sequences and Time Series Data.* PhD thesis, University of Minnesota, 2009.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, September, 2009.

[10] F.Y. Edgeworth. On discordant observations. *Philosophical Magazine*, pages 364–375, 1887.

[11] Siong Thye Goh and Cynthia Rudin. Box drawings for learning with imbalanced data. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 333–342, 2014.

[12] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24:1641–1650, 2003.

[13] V.J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, pages 85–126, 2004.

[14] R.V. Hogg. Statistical robustness: One view of its use in applications today. *American Statistics*, 33:108–116, 1979.

[15] Woods Hole Oceanographic Institute. Remus. http://www.whoi.edu/main/remus. Accessed: 2015-02-24.

[16] Kongsberg Maritime. Autonomous underwater vehicles - AUVs. www.km.kongsberg.com. Accessed: 2015-02-24.

[17] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

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

[19] Emmanuel Parzen. On the estimation of a probability density function and the mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[20] Bernhard Scholkopf, Robert Williamsonx, Alex Smolax, John Shawe-Taylory, and John Platt. Support vector method for novelty detection. *NIPS*, 12:582–588, 1999.

[21] David M.J. Tax and Robert P.W. Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.

[22] Andreas Theissler. *Detecting Anomalies in Multivariate Time Series from Automotive Systems.* PhD thesis, Brunel University, 2013.