Dealing With Faulty Data Via a Physics-Based Filtering Method

Ross Pellenberg
ross.pellenberg@uconn.edu

Follow this and additional works at: https://opencommons.uconn.edu/gs_theses

Recommended Citation
Pellenberg, Ross, "Dealing With Faulty Data Via a Physics-Based Filtering Method" (2020). Master's Theses. 1505.
https://opencommons.uconn.edu/gs_theses/1505

This work is brought to you for free and open access by the University of Connecticut Graduate School at OpenCommons@UConn. It has been accepted for inclusion in Master's Theses by an authorized administrator of OpenCommons@UConn. For more information, please contact opencommons@uconn.edu.
Dealing With Faulty Data Via a Physics-Based Filtering Method

Ross Merrill Pellenberg

B.S., University of Connecticut, 2018

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

At the

University of Connecticut

2020
# Table of Contents

i. List of Figures .................................................................................................................. vi

ii. List of Tables ................................................................................................................... x

iii. Nomenclature/Glossary .................................................................................................. xii

iv. Abstract ............................................................................................................................ xvii

1 Introduction .......................................................................................................................... 1

1.1 Gas Turbine Health Management .................................................................................. 1

1.2 Scope and Aim of the Thesis ......................................................................................... 7

1.3 Literature Review- Have Physics-Based Filters Been Used Before? ............................. 7

2 Methodology ....................................................................................................................... 14

2.1 Useul Machine Learning Background .......................................................................... 14

2.2 The C-MAPSS Dataset ................................................................................................. 39

2.3 Literature Review- Previous Studies Using C-MAPSS Dataset .................................. 47

2.4 Novel Contributions of This Thesis ............................................................................. 50

2.4.1 Second law of thermodynamics applied to the compressor ........................................ 50

2.4.2 Conservation of Energy: Combustor ......................................................................... 54

2.5 Preparing the Training Set and Training Machine Learning Models ........................... 58

2.6 Preparing the Testing Sets and Testing the Trained Models ......................................... 65

2.7 Evaluating Algorithm Performance ............................................................................. 68

3 Results and Discussion ...................................................................................................... 73

3.1 Training Set Results ..................................................................................................... 73

3.2 Testing Set Results ....................................................................................................... 90

3.2.1 Test #1: Test On Good Data ...................................................................................... 91

3.2.2 Test #2: Test With 1000 Harshly Faulty Data Points in Testing Set ......................... 100

3.2.3 Test #3: Test With 1000 Harshly Faulty Data Points Using Physics-Based Filter .... 106

3.2.4 Test #4: Test With 1000 Mildly Faulty Data Points in Testing Set ............................ 113

3.2.5 Test #5: Test With 1000 Mildly Faulty Data Points Using Physics-Based Filter .... 119

3.2.6 Test #6: Test With Even-Numbered Engine Harsh Faults In Testing Set ................. 126

3.2.7 Test #7: Test With Even-Numbered Engine Harsh Faults Using Physics-Based Filter ......................................................................................................................... 132

3.2.8 Test #8: Test With Even-Numbered Engine Mild Faults In Testing Set ................. 138
3.2.9 Test #9: Test With Even-Numbered Engine Mild Faults Using Physics-Based Filter
........................................................................................................................................ 144

4 Conclusions .......................................................................................................................................... 148

4.1 Summary ............................................................................................................................................. 148

4.2 Future Work ......................................................................................................................................... 149

5 References ........................................................................................................................................... 151
i. List of Figures

Figure 1-1: Pratt & Whitney 4084 Gas Turbine Engine in Ref. [12] ......................................................... 1
Figure 1-2: Example of an Instrumentation Schematic for a Single Stage Compression and
Figure 1-3: Scopus Key Word Search “Gas Turbine” AND Predictive ................................................. 5
Figure 1-4: Artificial Neural Network Structure and Sample of Training and Test Data Used in
Ref. [36] ................................................................................................................................................. 9
Figure 1-5: Comparison of Absolute Error (L1 Norm) Between Prediction and True Solution for
Maximum Cell Temperature in Ref. [42] ............................................................................................ 11
Figure 2-1: Examples of Classification and Regression Problems in Ref. [22] ................................. 16
Figure 2-2: Example Visualization of Gradient Descent in Ref. [32] .............................................. 18
Figure 2-3: Graph of the Sigmoid Function Used to Express the Hypothesis in Logistic
Regression ........................................................................................................................................... 22
Figure 2-4: Plot of the Two Cost Functions for Logistic Regression .................................................. 24
Figure 2-5: Examples of Different Fits for a Binary Classification Logistic Regression Problem
in Ref. [23] ........................................................................................................................................... 26
Figure 2-6: Examples of a Machine Learning Algorithm with High Bias (left) and High Variance
(right) in Ref. [21] .................................................................................................................................. 30
Figure 2-7: Cross Validation and Training Errors as a function of \( \lambda \) in Ref. [24] ....................... 31
Figure 2-8: Logistic Regression Cost Function (blue) and SVM Hinge Loss Cost Function
(orange) When \( y=0 \) and \( y=1 \) ............................................................................................................. 34
Figure 2-9: Schematic and Flowchart of Gas Turbine Engine Modeled in C-MAPSS Software in
Ref. [30] .................................................................................................................................................. 39
Figure 2-10: Response Surfaces of Sensed Outputs from Cycle Deck in Ref. [9] ......................... 40
Figure 2-11: Operational Margin of HPC Fault in Cycle Deck in Ref. [9] ............................................ 41
Figure 2-12: Breakdown of C-MAPSS Datasets Usage in Ref. [14] .................................................. 47
Figure 2-13: Diagram of Compressor Showing the Inlet, Outlet, Control Volume, and Work
Input in Ref. [27] .................................................................................................................................... 51
Figure 2-14: Diagram of Combustor and Turbine (left) and Turbine Showing the Inlet, Outlet,
Control Volume, and Work Output (right) in Refs. [81][27] .............................................................. 55
Figure 2-15: 3D Plot and 2D slice of F1 Score Vs. Precision and Recall [29] ................................. 71
Figure 3-1: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered,
Unnormalized, Multiclass Training Set ............................................................................................... 73
Figure 3-2: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered,
Unnormalized, Binary Training Set ..................................................................................................... 74
Figure 3-3: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered,
Normalized, Multiclass Training Set ................................................................................................. 77
Figure 3-4: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered,
Normalized, Binary Training Set ....................................................................................................... 78
Figure 3-5: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Filtered,
Unnormalized, Multiclass Training Set ............................................................................................. 81
Figure 3-25: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 109
Figure 3-26: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 110
Figure 3-27: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples .................................................. 114
Figure 3-28: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples .................................................. 114
Figure 3-29: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples .................................................. 115
Figure 3-30: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples .................................................. 115
Figure 3-31: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with 1000 Mildly Faulty Examples .................................................. 116
Figure 3-32: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 121
Figure 3-33: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 122
Figure 3-34: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 122
Figure 3-35: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 123
Figure 3-36: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter ......................................................... 123
Figure 3-37: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Harsh Faults on Even-numbered Engines .......... 127
Figure 3-38: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Harsh Faults on Even-numbered Engines .......... 128
Figure 3-39: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Harsh Faults on Even-numbered Engines .......... 128
Figure 3-40: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Harsh Faults on Even-numbered Engines .......... 129
Figure 3-41: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Harsh Faults on Even-numbered Engines .......... 129
Figure 3-42: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter............................................................................................................................................. 134
Figure 3-43: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter............................................................................................................................................. 134
Figure 3-44: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter............................................................................................................................................. 135
Figure 3-45: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter............................................................................................................................................. 135
Figure 3-46: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter............................................................................................................................................. 136
Figure 3-47: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Mild Faults on Even-numbered Engines............. 139
Figure 3-48: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Mild Faults on Even-numbered Engines............. 140
Figure 3-49: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Mild Faults on Even-numbered Engines.................. 140
Figure 3-50: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Mild Faults on Even-numbered Engines.................. 141
Figure 3-51: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Mild Faults on Even-numbered Engines ....................... 141
ii. **List of Tables**

Table 2-1: Input Parameters to C-MAPSS Engine Simulation in Ref. [30] .................................................. 45
Table 2-2: Sensed Outputs from C-MAPSS Engine Simulation in Ref. [30] .................................................. 45
Table 2-3: Overview of the C-MAPSS Datasets in Ref. [14] ................................................................................ 46
Table 2-4: Breakdown of Problem Framing Using C-MAPSS Datasets in Ref. [14] ................................. 48
Table 2-5: Breakdown of Classification Methods Using C-MAPSS Datasets in Ref. [14] .............. 48
Table 2-6: Breakdown of Machine Learning Methods Used on C-MAPSS Datasets in Ref. [14] ......................... 49
Table 2-7: Breakdown of Training Set Multiclass Classification ................................................................. 61
Table 2-8: Breakdown of Training Set Binary Classification ............................................................................. 62
Table 2-9: Breakdown of 10 Engine Training Set Multiclass Classification .............................................. 62
Table 2-10: Breakdown of Training Set Binary Classification ........................................................................ 62
Table 2-11: Matrix of Trained SVM Models .................................................................................................... 63
Table 2-12: Breakdown of Testing Set Multiclass Classification ................................................................. 66
Table 2-13: Breakdown of Testing Set Binary Classification ........................................................................... 66
Table 3-1: Summary of Training Set Results For Models Trained on Unfiltered, Unnormalized Data .................................................................................................................. 75
Table 3-2: Summary of Training Set Results For Models Trained on Unfiltered, Normalized Data ........................................................................................................................................ 79
Table 3-3: Summary of Training Set Results For Models Trained on Filtered, Unnormalized Data ........................................................................................................................................ 83
Table 3-4: Summary of Training Set Results for Models Trained on Filtered, Normalized Data 87
Table 3-5: Summary of the Best Performing Trained Models Performance on the Training Sets88
Table 3-6: Summary of Results for All Well-Performing Trained Models on the Testing Set From Test #1 ......................................................................................................................................... 97
Table 3-7: Summary of Trained Models That Demonstrate Adequate Performance on the Good Data Testing Set from Test #1 ............................................................................................................. 99
Table 3-8: Summary of Model Performance on Testing Set with 1000 Harshly Faulty Examples Compared To Performance on Good Data .......................................................................................................................... 105
Table 3-9: Summary of Classification Distribution of Unfiltered Multiclass Testing Set with 1000 Harshly Faulty Data Points Removed ............................................................................................................. 107
Table 3-10: Summary of Classification Distribution of Unfiltered Binary Testing Set with 1000 Harshly Faulty Data Points Removed ............................................................................................................. 107
Table 3-11: Summary of Classification Distribution of Filtered Multiclass Testing Set with 1000 Harshly Faulty Data Points Removed ............................................................................................................. 107
Table 3-12: Summary of Classification Distribution of Filtered Binary Testing Set with 1000 Harshly Faulty Data Points Removed ............................................................................................................. 107
Table 3-13: Summary of Model Performance on Testing Set with 1000 Harshly Faulty Examples Ran Through the Physics-based Filter Compared to Performance on Testing Set with 1000 Harshly Faulty Examples and Performance on Good Data .......................................................................................................................... 112
Table 3-14: Summary of Model Performance on Testing Set with 1000 Mildly Faulty Examples Compared to Performance on Testing Set with 1000 Harshly Faulty Examples and Performance on Good Data

Table 3-15: Summary of Classification Distribution of Unfiltered Multiclass Testing Set with 1000 Mildly Faulty Data Points Removed

Table 3-16: Summary of Classification Distribution of Unfiltered Binary Testing Set with 1000 Mildly Faulty Data Points Removed

Table 3-17: Summary of Classification Distribution of Filtered Multiclass Testing Set with 1000 Mildly Faulty Data Points Removed

Table 3-18: Summary of Classification Distribution of Filtered Binary Testing Set with 1000 Mildly Faulty Data Points Removed

Table 3-19: Summary of Model Performance on Testing Set with 1000 Mildly Faulty Examples Ran Through Physics Filter Compared to Performance on Testing Set with 1000 Mildly Faulty Examples and Performance on Good Data

Table 3-20: Summary of Model Performance on Testing Set with Even Engine Harsh Faults Compared to Performance on Good Data

Table 3-21: Summary of Classification Distribution of Multiclass Testing Set with Even Engine Faulty Data Points Removed

Table 3-22: Summary of Classification Distribution of Binary Testing Set with Even Engine Faulty Data Points Removed

Table 3-23: Summary of Model Performance on Testing Set with Even Engine Harsh Faults Ran Through Physics Filter Compared with Performance on Testing Set with Even Engine Harsh Faults and Testing Set with Good Data

Table 3-24: Summary of Model Performance on Testing Set with Even Engine Mild Faults Exam Compared to Performance on Testing Set with Even Engine Harsh Faults and Performance on Good Data

Table 3-25: Summary of Model Performance on Testing Set with Even Engine Mild Faults Ran Through Physics Filter Compared with Performance on Testing Set with Even Engine Mild Faults and Testing Set with Good Data
### Nomenclature/Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC</td>
<td>Low Pressure Compressor</td>
</tr>
<tr>
<td>HPC</td>
<td>High Pressure Compressor</td>
</tr>
<tr>
<td>HPT</td>
<td>High Pressure Turbine</td>
</tr>
<tr>
<td>LPT</td>
<td>Low Pressure Turbine</td>
</tr>
<tr>
<td>EHM</td>
<td>Equipment Health Management</td>
</tr>
<tr>
<td>SME</td>
<td>Subject Matter Expert</td>
</tr>
<tr>
<td>RUL</td>
<td>Remaining Useful Life</td>
</tr>
<tr>
<td>APR</td>
<td>Advanced Pattern Recognition</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>BDS</td>
<td>Bad Data Suppression</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>C-MAPSS</td>
<td>Commercial-Modular Aero-Propulsion System Simulations</td>
</tr>
<tr>
<td>PHM</td>
<td>Prognostics and Health Management</td>
</tr>
<tr>
<td>LHV</td>
<td>Lower Heating Value</td>
</tr>
<tr>
<td>DAQ</td>
<td>Data Acquisition</td>
</tr>
</tbody>
</table>
SMA  Simple Moving Average
PCA  Principal Component Analysis
TTF  Time To Failure
CV  Cross Validation
θ  Learned Parameters of Machine Learning Model
x  Input Variables to Machine Learning Model
h_\theta(x)  Hypothesis
y  Output(s) of Machine Learning Model
α  Learning Rate
m  Number of Examples in a Dataset
n  Number of Variables in a Dataset
J(\theta)  Machine Learning Cost Function
\frac{\partial}{\partial \theta_j} J(\theta)  Gradient of Cost Function
\mu_i  Mean of a Variable
s_i  Standard Deviation of a Variable
g(z)  Sigmoid Function
C(x, y)  Combination Operator
λ  Regularization Parameter
C  SVM Regularization Parameter

cost_0  Cost Function when \( y = 0 \)

\( \text{cost}_1 \)  Cost Function when \( y = 1 \)

l  Landmark

\( f_i \)  New Variable After Applying Kernel Function to \( x_i \)

\( \sigma^2 \)  SVM Bandwidth Parameter

F  Vector of \( f_i \) values

w  Wear

h(t)  Engine Component Health Index as Function of Time

\( th_w \)  Upper Wear Threshold

d  Initial Degradation

e(t)  Efficiency Health Index as Function of Time

f(t)  Flow Health Index as Function of Time

H(t)  Overall Engine Health Index

\( m_x \)  Current Position Relative to Operability Margin for Component \( x \)

\( \delta q \)  Differential Thermal Energy

\( \delta w \)  Differential Work

du  Differential Internal Energy
ds  Differential Entropy

dv  Differential Specific Volume

dT  Differential Temperature

dP  Differential Pressure

R   Specific Universal Gas Constant

C_p Specific Heat at Constant Pressure

C_v Specific Heat at Constant Volume

T   Temperature

T_s Isentropic Temperature

T_t Total Temperature

P   Pressure

P_t Total Pressure

γ   Ratio of Specific Heats

CPR Compressor Pressure Ratio

TPR Turbine Pressure Ratio

\dot{W}_c Compressor Work

\dot{W}_t Turbine Work

\eta_c Compressor Adiabatic Efficiency
η_t \quad \text{Turbine Adiabatic Efficiency}

\dot{Q} \quad \text{Heat Exchanged}

\dot{m}_f \quad \text{Fuel Mass Flow Rate}

\dot{m}_{\text{air}} \quad \text{Air Mass Flow Rate}

\phi \quad \text{Fuel-Air Ratio}

\Delta h_{\text{comb}} \quad \text{Heat of Combustion}

Pr \quad \text{Precision}

Re \quad \text{Recall}
iv. Abstract

Gas turbines are expensive, revenue generating machines and, as such, there is a strong interest in practicing state-of-the-art maintenance techniques to keep them running at healthy performance levels. One way to monitor and evaluate gas turbine health is to train a machine learning model on historical run-to-failure sensor data to differentiate between healthy and unhealthy performance. The biggest barrier to building these models is the scarcity of run-to-failure data. Not only is this data expensive and time consuming to acquire, but the data is often not publicly released for competitive purposes. This thesis uses a publicly available run-to-failure dataset previously created through the C-MAPSS gas turbine engine simulation software to train and test Support Vector Machine (SVM) machine learning algorithms that classify the health of a gas turbine engine. In particular, this thesis studies the performance of the models when they are asked to make predictions on a new dataset containing faulty data; defined as data that violate the laws of thermodynamics. Faulty data can realistically be acquired if sensors aren’t calibrated and/or the wrong type of sensor is used to measure a certain parameter. A novel physics-based filter is introduced that scans the dataset and removes all instances of violations of thermodynamic laws. The ability of the physics-based filter to identify remove faulty data is highly dependent on the signal processing steps taken, if any, before applying the filter. The trained machine learning models are tested on new datasets where the faulty data has either been left in, or some/all of it has been removed using the physics-based filter. By testing scenarios in which the physics-based filter is applied either before or after signal processing occurs, the importance of evaluating the quality of a dataset before using it for analysis is illuminated.
1 Introduction
1.1 Gas Turbine Health Management

A gas turbine is a category of combustion engine that converts energy from burning fuels, such as natural gas, to mechanical energy [3]. This mechanical energy can be used as thrust for a plane or rocket, or it can drive a generator that produces electrical energy at a power plant. Industry standard gas turbines consist of five major components: The low-pressure compressor (LPC), high-pressure compressor (HPC), combustor, high-pressure turbine (HPT), and the low-pressure turbine (LPT). Air flows into the compressor section where it is compressed and then mixed with fuel and ignited in the combustor section. This hot gas is then used to spin the turbine blades from which work is extracted either in the form of thrust for a plane or energy to drive a generator. Figure 1-1 below shows a schematic of a Pratt & Whitney 4084 gas turbine engine.

![Figure 1-1: Pratt & Whitney 4084 Gas Turbine Engine in Ref. [12]](image-url)
Gas turbines are expensive, revenue generating machines and, as such, there is a strong interest in practicing state-of-the-art maintenance techniques to keep them running at healthy performance levels. This leads into a discussion of the widespread practice of Equipment Health Management (EHM), which can be broken down into three categories: monitoring, diagnostics, and recommendations [1][2]. Monitoring involves instrumenting the gas turbine with sensors and creating a historical record of these sensor measurements. Typical gas turbine sensors include accelerometers, thermocouples, and pressure transducers. Some notable performance parameters that are measured with sensors are compressor and turbine inlet and outlet pressures and temperatures, as well as bearing vibrations. Figure 1-2 shows instrumentation that is typically found on gas turbines. It is important to note that this Figure 1-2 shows a single stage compression and multistage expansion gas turbine, while the gas turbine studied in thesis is multistage compression and expansion. The typical instrumentation is similar, regardless.

![Figure 1-2: Example of an Instrumentation Schematic for a Single Stage Compression and Multistage Expansion Gas Turbine in Ref. [13]](image-url)
Monitoring can be done both on-site and remotely. On-site monitoring involves a field team physically going up to the machine and taking measurements. Remote monitoring involves the sensor signals getting automatically fed into a data server for an operating engineer to view at their desk.

The second component of EHM is diagnostics. When a problem is noticed while monitoring the turbine, engineers and/or an automated system work to pinpoint the cause of the problem.

The third piece, recommendation, is where the equipment owner/operator gets information on the next steps to take with the machine in question. This could be a statement on which component is causing the problem, or a plan on what corrective steps should be taken. Recommendation empowers the equipment owner to make an educated decision on how to proceed with the machine in question [1]. This thesis critically assesses the diagnosis step; an overview of the techniques commonly used for diagnosis is presented, along with an exploration of a novel method of sensor data analysis.

In order for diagnosis to occur, there must be a clear understanding of what a problem actually looks like [1]. In other words, how do sensor readings change when the machine changes state from healthy to unhealthy? Answering this question has historically involved direct input from Subject Matter Experts (SMEs). SMEs in this case are typically the engineers that played a direct role in the design and commissioning of the gas turbine unit. Given their knowledge and familiarity with the machine, SMEs are the go-to source to troubleshoot any problems that arise. While SMEs are extremely useful tools for engine maintenance, they are only human. Their knowledge is contained in their head, and while they may try to pass their learnings on, there is always a risk of important information being lost forever. Using SMEs as
the primary tool to diagnose engine faults leads to performing scheduled maintenance. Scheduled maintenance involves shutting down the turbine and performing inspections and overhauls after a specific usage milestone is reached (i.e., 20,000 cycles). However, this is not ideal because there is a chance that the engine could have continued to perform in a healthy manner well passed the usage milestone, which causes loss of value for the customer. Instead of scheduled maintenance, a predictive maintenance scheme can be implemented if a suitable method were to be developed that could assess the state of the machine and estimate how many remaining cycles it has left in its healthy state. Furthermore, unforeseen faults arising leads to unscheduled maintenance. If an unexpected fault occurs before the usage milestone is reached, the machine needs to be shut down early, which can be very costly [7]. As such, there has been a strong push over the last half century to move towards computer-assisted and even entirely automated methods to use sensor data to diagnose engine faults. A Scopus key word search using “gas turbine” and “predictive” shows that papers started being written on this topic in the early 1970’s and Figure 1-3 below shows that interest in the field has steadily grown since.
The key idea of diagnosis is that it identifies a problem that has already occurred. This idea, along with the increasing interest in predictive maintenance paves the way for a stage in between monitoring and diagnostics: *prognosis*. Prognosis involves predicting the future state of the equipment via an analysis of the data gathered from monitoring. This future state could be a prediction of component failure from data trends, or an estimated Remaining Useful Life (RUL) of the machine. To that end, advanced pattern recognition (APR) techniques, including machine learning, are implemented in order to highlight degradation patterns in data that have previously gone unnoticed by the human eye [4].

Prognosis of gas turbines faults can come from high-fidelity, realistic, physics-based models of the specific gas turbine [6-8]. The issue with creating these models is that they are time consuming and expensive, and the cost that goes into building them may outweigh the savings achieved from predicting faults [9]. As such, these models are typically reserved for components whose failure would be catastrophic and result in complete system failure. An
example of a situation that necessitates realistic turbine modeling is launching a manned rocket into space, as it is imperative to ensure that the rocket will be safe enough to facilitate survival of the crew. Prognosis of typical wear-and-tear gas turbine faults, such as compressor fouling and bearing degradation, are better suited for data-driven models built on historical sensor data. These models are then applied to real-time incoming sensor data in order to identify any faults that may be present in the system. Some common models are based on gas path analysis [5] and anomaly detection via degradation trends [10]. The biggest barrier when building these models is the scarcity of sensor data and, more specifically scarcity of failure data [11]. This is because acquiring this data is another time consuming and expensive task. Depending on the instrumentation, gas turbines can produce thousands of measurements per second, much of which is not interesting during steady state operation. Systems that can handle this sort of data collection are also expensive and time consuming to configure and install. Furthermore, companies with these systems have an incentive to keep the data to themselves for competitive purposes. As such, finding a publicly available and physically accurate dataset describing the evolution of a gas turbine from a healthy state to a failure state is extremely important for building a prognostic model. The requirements to find a suitable dataset will be discussed in the next chapter of this thesis.

Due to the scarcity of run-to-failure data, it is desirable to implement a machine learning model(s) on any suitable datasets available, as once the model is trained and tested to satisfaction, it should work well on any new dataset given to it. This thesis explores the response of a machine learning algorithm, previously trained on a suitable run-to-failure dataset, when it is asked to make predictions on a new dataset containing faulty data. The machine learning algorithm aims to classify the health of a gas turbine engine using sensor data gathered over
individual flights. As such, bad data can realistically be acquired if sensors aren’t calibrated and/or the wrong type of sensor is used to measure a certain parameter. For example, if a K-type thermocouple calibration is mistakenly used to read measurements from a T-type thermocouple, the collected data will be incorrect.

1.2 Scope and Aim of the Thesis

In this thesis, we define faulty data as data that violates the laws of thermodynamics. A physics-based filter is introduced that scans the dataset and removes all instances of violations of the thermodynamic laws. The trained machine learning models are then tested on new datasets where the faulty data has either been left in, or some/all of it has been removed using the physics-based filter. This thesis hypothesizes that the physics-based filter will allow the trained models to perform just as well on bad datasets as they do on good datasets, while also alerting any engineer responsible for the data that there is a sensor issue.

Chapter 2 of this thesis, Methodology, will provide the necessary background on Machine Learning, the origin of the datasets used to train and test the chosen models, as well as the novel contributions. The Methodology chapter also discusses the makeup of the training and testing datasets, how they are processed for use, and how performance is evaluated for the machine learning models. Chapter 3, Results and Discussion, presents the successfully trained models and explores how they perform when asked to make predictions on faulty data. Finally Chapter 4, Conclusion, summarizes the previous chapters and provides guidance on future work based on the findings in this thesis.

1.3 Literature Review- Have Physics-Based Filters Been Used Before?

As previously explained, prognosis of typical wear-and-tear gas turbine faults are well-suited for data-driven models. Sensor data is used to predict the current and future states of the
engine, which negates the need to spend large amounts of time and money building realistic, physics-based models of the machine. One consequence of using a purely data-driven model, though, is the negative affect that faulty data can have on the predictive performance of the model. Further, data-driven models that give no consideration to the underlying physics may miss a significant part of the picture. This section presents a brief review on techniques used to deal with faulty data both in the gas turbine health management research field, as well as an exploration of physics-informed machine learning techniques.

In searching the literature using specific terms such as “bad data gas turbine” and “bad data machine learning,” published articles using data-driven approaches to identify bad data are returned. In 1999, Depold and Gass suggested that improvements to current health management practices including “application of statistical analysis and artificial neural network filters to improve data quality” [34]. Since then, there have been some efforts to apply these data-driven approaches to evaluating sensor data.

Ogaji et. al. describes the use of Artificial Neural Networks (ANNs), a type of machine learning algorithm, to diagnose single and dual sensor failures in a two-shaft stationary gas turbine [35]. They trained 3 ANNs, one to categorize sensor measurements as having faults or not having faults, a second to classify whether the fault is a sensor or component fault, and a third to identify the faulty sensor(s) and quantify the magnitude of the fault. This type of approach is useful for when faulty data does not violate thermodynamic laws and is instead only slightly different than the not faulty data.

Palme et. al. also employs ANNs for a slightly different purpose: sensor validation [36]. Their goal was to propose a method to identify failing sensors by training an ANN classifier to identify “sensor drift.” The purpose of this study is to reduce engine down-time caused by the
need to recalibrate/replace sensors that are no longer working well. Figure 1-4 below shows the ANN structure as well as a sample of data used to train and test the ANN models.

![Artificial Neural Network Structure and Sample of Training and Test Data Used in Ref. [36]](image)

Various sensor inputs are fed into the classification ANN whose outputs indicate whether the sensors are healthy or if they are experiencing sensor drift in a particular direction. They found that ANNs can indeed identify these drifts and, as such, can mitigate engine down time and costs associated with performing maintenance on the sensors.

Shi et. al. uses a different machine learning approach, constraint-based learning [38], to detect and adapt to sensor failures [37]. They utilized historical sensor data to formulate relationships between sensors and define constraints. Detection of sensor failure is indicated when a constraint is broken, and the established constraints are used to estimate what the real sensor values should be. This type of approach is useful when plenty of historical sensor data is available, although this is not always be the case.

Swischuk et. al. propose another data-driven approach to detect sensor failures and predict correct sensor data using autocorrelation and k-nearest neighbors regression [39][40][41]. Autocorrelation is used to classify the state of the sensor as failing or not failing. K-nearest
neighbors regression is used to predict the correct sensor values of a sensors determined to be failing. This approach is dependent on the availability of redundant sensor output which is not always be available in the field.

In addition to the data-driven gas turbine sensor data evaluation methods discussed above, physics-informed machine learning is a fast-growing field that is highly relevant to this thesis. At Pacific Northwest National Laboratory, Bao et. al. applied physics-informed machine learning to a solid oxide fuel cell system. They used variations of physics-informed machine learning methods and compared the predictions to those of the traditional, data-driven machine learning method and a Kriging-based surrogate model [42]. One variation involved learning the errors between the Kriging prediction on the true solution and adding the learned error term to the Kriging prediction. Figure 1-5 below shows results from the first approach.
As the number of training data increase, the physics-informed model experiences less predictive error compared to the true solution than the other models. A second method utilized deep neural networks and the mass balance model “to significantly decrease the reduced order model (ROM) prediction error with fewer training data compared to the traditional Kriging-based” predictions. Another approach used deep neural networks regression, the mass balance model, and a neural network classifier to classify the physical operating conditions of a natural gas fuel cell. Ultimately, all physics-informed methods provide improved performance compared to their data-driven counterparts.

Raissi et. al. use physics-informed machine learning, specifically physics-informed neural networks, to solve nonlinear partial differential equations [43]. They demonstrate how a data-driven approach using deep neural networks can approximate the solution to partial differential
equations such as the Burgers’ Equation [44] given sufficient training data. They go on to show how incorporating physics into the machine learning model (i.e., making sure any approximate solution satisfies the relevant physical laws) allows the solution to be approximated with less training data. This is a significant finding because lack of data is a prevalent problem in many research fields, which makes implementing purely data-driven methods difficult. The researchers coin this phenomenon as operating in the “small data regime” [43]. Further, these physics-informed neural networks are used to build both continuous and discrete time models. Overall, while data-driven methods are capable of approximating solutions to partial differential equations, incorporating physical laws makes these methods more efficient.

Das et al. employ dynamic mode decomposition (DMD), to develop a data-driven and physics-informed prognostic model to predict the remaining useful life of infrastructure components and systems [45]. They apply their model to predict mortar cube cracks under compressive load testing. Video is taken of the compressive load test and each frame is cropped and transformed into a matrix of values corresponding to individual pixels in the original frame. DMD is applied to data from these matrices to extract the dynamic modes of the nonlinear system which are then used to train the prognostic model. This method allows the nonlinear dynamics of crack propagation to be characterized without the need for a governing equation of the system; only the initial condition of the infrastructure needs to be approximated. The model is validated by using the initial condition to predict images of the crack propagation which is then compared to the images that the model is trained on.

Various use cases of machine learning for faulty sensor identification in gas turbine engines are presented in this section. Further, the concept of physics-informed machine learning is
introduced. This thesis aims to combine the two to implement a physics-informed method of faulty sensor data identification and explore its effect on machine learning model performance.
2 Methodology
2.1 Useful Machine Learning Background

Machine Learning is defined as “a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty” [18]. These patterns, as previously mentioned, are typically hidden in such a way that an SME would not be able to identify them without the help of a computer. Machine learning can be broken down into categories: supervised and unsupervised learning [1]. Supervised learning is the case in which the dataset has known inputs and outputs. An example of a supervised learning case is giving the machine a set of emails marked as spam or not spam, and then asking it to learn the difference between the two and make predictions on a new set of emails given to it. Unsupervised learning is the case in which a dataset is given to the computer and the computer is asked to identify patterns it sees within the data and come up with its own relationships between the input variables. An example of an unsupervised learning case is giving a computer a dataset of 1,000,000 genes and asking it to group the genes based on similarities that the computer determines itself. Regardless of whether supervised or unsupervised learning is desired, the data given to the computer should be partitioned in the same way. First, the algorithm is trained with a training set which is made up of a portion of the entire dataset. This training set consists of $m$ training examples, with each example having its own input(s) and, in the case of supervised learning, output(s). The algorithm learns based on the training set and is then tested with the other portion of the dataset called the testing set. An optional step is to include a validation dataset, which is a dataset different from the training/testing set that is used to validate the performance of the machine learning algorithm against new data. Whether or not a separate validation set is used (or even available), the purpose of portioning the dataset into a testing and training set is to validate the algorithm. This
is typically done in one of two ways: holdout validation or cross fold validation [19]. Holdout validation is when a pre-determined percentage of the dataset is withheld at random from the training set. The algorithm then fits a function to the training set and uses the portion that was held out in order to test the learned model and evaluate its performance. Cross validation is “when the data is divided into \( k \) subsets, and the holdout method is repeated \( k \) times” [19]. With this method, each of the \( k \) subsets is used once as the testing set, while the other \( k - 1 \) subsets makeup the training set. After holdout validation is performed \( k \) times, the average error of all \( k \) trials is calculated. The advantage to this method is that every single data point in the dataset is used in exactly one training set and is then in the training set during the other \( k - 1 \) learning periods.

There are two types of problems that supervised machine learning can be applied to: classification and regression [21]. Classification asks the algorithm to identify outputs as discrete, fixed values. For example, [2] asks a machine learning algorithm to identify whether the data it is given is associated with one of two possible valves installed on a gas turbine. Regression asks the algorithm to predict a continuous valued output. For example a machine learning algorithm can be asked to predict future sale prices of houses on the market after being given a historical dataset of house sizes and their corresponding sale prices. Figure 2-1 below shows the difference between classification and regression.
Classification asks the computer to fit the parameters of a function that describes the decision boundary (the black curve in the graph on the left) between two or more classes. The decision boundary is just that, a curve that tries to make a decision on which output class an input belongs to based on where the input falls relative to the boundary. Regression, on the other hand, fits a curve to a dataset and explicitly passes new inputs into that function to predict what the output will be. Three of the most popular machine learning algorithms are linear regression, logistic regression, and support vector machines. The following discussion gives some background and mathematical details on each of these algorithms.

In general, the following problem is posed that is suitable for a supervised machine learning algorithm: \textit{given a training set, learn a function }\( h: X \rightarrow Y \)\textit{ such that }\( h(X) \)\textit{ is a “good” predictor for the corresponding value of }\( y \) [21]. Here, \( h \) is the hypothesis that maps x’s (inputs) to y’s (outputs). For univariate linear regression, the hypothesis is defined as \( h_{\theta}(x) = \theta_0 + \theta_1 x \) where \( \theta_0 \) and \( \theta_1 \) are parameters of the model. The idea is to choose the parameters such that \( h_{\theta}(x) \) is
close to the output y for every training example \((x, y)\). In order to do this, the cost function is defined as the sum of squared errors between the hypothesis and the set of outputs for \(m\) training example as follows:

\[
J(\theta_0 + \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2
\]

(1)

The goal is to choose the theta parameters such that this cost function is minimized. A popular method for minimization is called gradient descent. The idea behind gradient descent is to start with some initial guess for \(\theta_0\) and \(\theta_1\), and then simultaneously change each of them to reduce the cost, \(J(\theta_0 + \theta_1)\), until a local minimum is reached. The gradient descent algorithm can be written as follows:

Repeat until convergence

\[
\begin{align*}
\theta_j &:= \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_0 + \theta_1) \quad \text{for } j = 0 \text{ and } j = 1
\end{align*}
\]

where \(\alpha\) is the learning rate, a scaling factor that controls how large the steps that gradient descent are.

\[
\frac{d}{d\theta_j} J(\theta_0 + \theta_1), \text{ the gradient of the cost function, is defined as:}
\]
\[
\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})
\]

(2)

\[
\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}
\]

(3)

This implementation is sometimes referred to as “batch” gradient descent because each training example is used during each iteration of the algorithm. A visualization of gradient descent is shown in Figure 2-2 below.

Figure 2-2: Example Visualization of Gradient Descent in Ref. [32]

In Figure 2-2, the rightmost red circle shows the initial guesses for the parameters. Each black star shows how one iteration of gradient descent changed the value of these parameters. Ultimately, a local minimum of the cost function is found at the rightmost red arrow. Notice that had gradient descent been initialized at the leftmost red circle, a different local minimum for the
cost function would have been reached. Care should be taken when in choosing the learning rate, as a large $\alpha$ can cause gradient descent to overshoot the minimum and fail to converge or even diverge. A small $\alpha$ will cause gradient descent to run slowly, but it will still converge to a local minimum. The best way to ensure that gradient descent is implemented correctly is to plot the cost function versus iterations of gradient descent and check that the value of the cost function decreases on each iteration. If that is not the case, then $\alpha$ is too large [21].

Both linear regression and gradient descent can be generalized to multiple variables. For multivariate linear regression with $n$ variables, the hypothesis is written as

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$$

(4)

Where the $T$ is the transpose operator applied to a vector of $\theta$ parameters. In this equation, $x_0 = 1$ always (notice how $x_0$ was omitted from the hypothesis in univariate linear regression). It follows that gradient descent will update each theta parameter in the same way as in linear regression.

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

(5)

When using gradient descent, it is important that all the input variables are on a similar scale. For example, linear regression with gradient descent can be implemented to estimate the sale
price of a house given the property size in square feet and the number of bedrooms. Notice that
the property size will likely be in the thousands and the number of bedrooms will be less than 10.
This kind of discrepancy in scale between input variables can cause gradient descent to run very
slowly and it may not converge. A common way to deal with this is called feature scaling.
Feature scaling can be implemented in a few different ways but two of the most popular will be
discussed here. One implementation is simply dividing an instance of a variable by the
maximum instance of that variable. If the largest property size in the dataset is 2000 square feet,
then every variable \( x_i \) corresponding to a property size should be divided by 2000. Similarly, if
the largest number of bedrooms is 10, then every variable \( x_i \) corresponding to a number of
bedrooms in a house should be divided by 10. On the other hand, mean normalization takes a
variable \( x \) from a dataset and replaces that variable with a new one defined as

where \( \mu_i \) is the mean of the variable \( x \) and \( s \) is the standard deviation of the variable \( x \). Both of
the methods discussed help to ensure that all the input variables have similar ranges of values,
which allow gradient descent to run issue free, assuming it is implemented correctly as discussed
above [21].

Linear regression can be further generalized into polynomial regression if it is observed
that a nonlinear function would better fit the given dataset. This can be done by simply creating
new variables in the dataset that take other variables and raise them to certain powers. For
example, if property size is the only variable given to estimate the price of the house, one could
define their hypothesis as follows.
\[ h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \]  
\[ = \theta_0 + \theta_1 \cdot (\text{size}) + \theta_2 \cdot (\text{size})^2 + \theta_3 \cdot (\text{size})^3 \]  

With polynomial regression, feature scaling becomes extremely important as raising a large or very small variable to a power will only serve to make it much larger or much smaller. Linear regression works well for regression problems but is not recommended for classification problems. The next machine learning algorithm I will discuss is called logistic regression, which fits a function to create a decision boundary in classification problems. The decision boundary is a function that maps input variable(s) to one of two or more output classes.

Logistic regression is the most popular machine learning algorithm for binary classification problems. These are problems where the output falls into one of two classes, and it can also be extended to multiclass classification. An example of a problem suitable for logistic regression is predicting whether or not a tumor is malignant based on its size. In binary logistic regression, the output, \( y \), takes either the value 0 or 1. A 0 value corresponds to the negative class and a 1 value corresponds to the positive class. We want to formulate our problem such that \( 0 \leq h_\theta(x) \leq 1 \) and can be interpreted as the estimated probability that \( y=1 \) on input \( x \). This is mathematically expressed as \( h_\theta(x) = P(y = 1 \mid x; \theta) \) which translates to "the probability that \( y=1 \), given \( x \) parameterized by \( \theta \). Logistic regression gets its name because of the logistic function (also known as the sigmoid function) is used to describe the hypothesis. The logistic function is

\[ g(z) = \frac{1}{1 + e^{-z}} \]  

which is shown in Figure 2-3 below.
As is evident from the graph, the sigmoid function follows $g(z) \to 1$ as $z \to \infty$ and $g(z) \to 0$ as $z \to -\infty$. This is a valuable property as it allows the user to impose parameters, $\theta$, on the logistic function to estimate the probability that the input fits into one of two or more classes. This is mathematically expressed as $h_\theta(x) = g(\theta^T x) = P(y = 1 | x; \theta)$ where $T$ is the transpose operator applied to a vector of $\theta$ parameters. In long form this is written as

$$h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n) \quad (9)$$

Logistic regression can be applied to create both linear and non-linear decision boundaries which makes it a valuable machine learning algorithm for classification problems. Since the hypothesis estimates the probability that an input $x$ belongs to a certain class, it is fair for the algorithm to place an input into a certain output class if that input’s hypothesis of being in that class is either greater than 0.5 (binary classification) or the highest when compared to the other hypotheses (multiclass classification). This observation is the basis of determining the
decision boundary. From Figure 6, it is seen that the sigmoid function has a value of 0.5 when
it’s input $z = 0$, which allows a decision boundary to be formed around the area where this holds
true.

If a non-linear decision boundary is more suitable for the posed problem, then one can
simply introduce non-linear terms to the hypothesis. For example, if $h_\theta(x) = g(\theta_0 + \theta_1 x_1 +$
$\theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$ and the $\theta$ parameters are learned to be $[-1; 0; 0; 1; 1]$ then the algorithm
will predict $y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$ which can also be written as $x_1^2 + x_2^2 \geq 1$ which is the
equation of a circle centered on the origin with radius equal to 1 [21].

Up to this point, the discussion on logistic regression has been focused on the
mathematical formulation of the hypothesis and what a decision boundary can look like. The
most important part of logistic regression, as with any machine learning algorithm, is how the
computer learns the $\theta$ parameters. A suitable cost function for logistic regression needs to
penalize both false positives (incorrectly predicts the positive class) and false negatives
(incorrectly predicts the negative class). There are two functions that satisfy this criteria:

\[
\text{cost}(h_\theta(x), y) = -\log(h_\theta(x)) \text{ if } y = 1
\]

\[
\text{cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \text{ if } y = 0
\]

These functions are plotted in Figure 2-4 below.
Notice that the red line (representing when $y$ is actually equal to 0) goes to infinity (large penalty) as the hypothesis goes to 1 and that the blue line (representing when $y$ is actually 1) goes to infinity as the hypothesis goes to 0. This captures the intuition that the learning algorithm should be heavily penalized if $h_\theta(x) = 0$ but $y = 1$. It is possible to combine these two functions into one equation for cost as follows:

$$ cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) $$

This equation simplifies to the two shown above when $y = 1$ or $y = 0$. With the cost of a prediction of one training example defined, $f(\theta)$ can be written to evaluate the cost function over all training examples.
In order to fit the $\theta$ parameters, it is necessary to determine what values of $\theta$ minimize this cost function. This can be done using gradient descent just as with linear regression. It turns out that the only difference between implementing gradient descent for logistic regression and linear regression is that the hypothesis is formulated differently as has been discussed in this section.

The most common way to extend binary classification logistic regression to multiclass classification is by implementing the one vs. all method. Applying this method to $k$ classes involves training a logistic regression classifier $h_{\theta}^{(i)}(x)$ for each class $i$ to estimate the probability that $y = i$. In simple terms, binary classification logistic regression requires only one logistic regression classifier to predict which class of two classes an input belongs to. In multiclass classification with $K$ classes one needs to train $K$ classifiers, with each classifier comparing one class to all other examples lumped into another class [21]. An alternative to the one vs. all method is the one vs. one method [25]. Here, $C(K, 2)$, where $C(x, y)$ is the combination operator, logistic regression classifiers are trained in order to perform binary classification logistic regression between every possible combination of two output classes. For example, if one is trying to classify an input into one of three classes, one needs to train a logistic regression classifier for classes 1 and 2, classes 1 and 3, and classes 2 and 3. Clearly the one vs. one method is more computationally expensive for large classification problems, so care should be taken when choosing which method to implement. To make a classification prediction on a new input $x$, regardless of which method is chosen, the algorithm will choose the class $i$ that maximizes $h_{\theta}^{(i)}(x)$ [21].
When fitting the $\theta$ parameters it is possible to either underfit or overfit the dataset. If there are too many input variables that the algorithm learns the hypothesis may fit the training dataset well but fail to generalize to new examples; this is referred to as overfitting. Examples of underfit, overfit, and a good fit are show in Figure 2-5 below.

![Diagram of different fits for a binary classification logistic regression problem](image)

Figure 2-5: Examples of Different Fits for a Binary Classification Logistic Regression Problem in Ref. [23]

There are two common ways to mitigate overfitting: reducing the number of variables in the training set or regularization. Reducing the number of features is typically a manual task; the researcher goes through the dataset and uses their knowledge of the system to eliminate variables that may not be that important. There are also model selection algorithms that are used to determine and eliminate variables that aren’t useful for learning the hypothesis, but those are outside of the scope of this thesis [21]. Regularization is a technique that allows the researcher to keep all the variables in their dataset while reducing the magnitude of the $\theta$ parameters in order to learn a “simpler” hypothesis. Regularization can be applied to both linear and logistic regression as well as many other machine learning algorithms like support vector machines which will be discussed shortly. Regularization is implemented by introducing $\lambda$, the
regularization parameter, to the cost function of a machine learning algorithm. $\lambda$ acts as a scaling factor to reduce the magnitude of the $\theta$ parameters thus controlling the tradeoff of fitting the training set and keeping the parameters small. It is important to note that $\theta_0$ is not regularized because, as previously discussed, $x_0$ is always equal to one [21].

For linear regression, the cost function is modified with the regularization term as follows:

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

(14)

Note that if $\lambda$ is too large, all $\theta$ parameters will be heavily penalized and all will be close to 0 which makes the hypothesis look like $h_\theta(x) = \theta_0$ which is not useful as this is just a horizontal line. It follows then that the gradient descent algorithm must be modified with the regularization term as well. Gradient descent becomes:

Repeat until convergence

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x^{(i)} + \frac{\lambda}{m} \theta_j$$

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

where
\[
\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x^{(i)} + \frac{\lambda}{m} \theta_j
\]  

(15)

Similarly, the cost function for logistic regression is modified with the regularization terms as follows:

\[
J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2
\]  

(16)

To summarize, regularization is a technique used to learn “simple” parameters that fit a dataset and can generalize well to new inputs. With today’s advanced curve fitting techniques, it is easy to fit a complex polynomial function to any dataset, but it is another task entirely to fit a curve that is a good predictor on new examples. Regularization is an easy to implement technique that aids in accomplishing this task.

With background given on some of the most popular machine learning algorithms, it is also useful to discuss how to diagnose problems that arise when implementing them. The two most common problems that machine learning algorithms suffer from are high variance (overfit) or high bias (underfit) when fitting to a dataset. Recall that Figure 2-5 shows what these kinds of fits may look like. To mitigate these problems the most popular routes to take are:

- Get more training examples
- Try smaller sets of features (variables)
• Try getting additional features
• Try adding polynomial features
• Try decreasing the regularization parameter $\lambda$
• Try increasing the regularization parameter $\lambda$

With these steps in mind, the next logical questions to ask are how to determine what problem your algorithm is suffering from and then how to decide which route to take to fix it. Recall the earlier discussion on cross validation. We can now define the error (without regularization) on the training set, cross validation set, and test set as follows:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})^2$$  \hspace{1cm} (17)$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$  \hspace{1cm} (18)$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$  \hspace{1cm} (19)$$

These definitions allow learning curves to be plotted, which are these errors as a function of the training set size $m$. Examples of learning curves are shown in Figure 2-6 below.
High bias is characterized by $J_{cv} \approx J_{train}$ as the training set size increases. Note that these plots will look similar whether $J_{cv}$ or $J_{test}$ is used. Further, the errors both stay relatively high as the training set size increases. This is because high bias implies that the hypothesis is too simple to capture the complexity of the data. As such, adding more training examples will not help improve a Machine Learning algorithm fit that suffers from high bias.

High variance is characterized by a gap between $J_{cv}$ and $J_{train}$ when both are plotted against the training set size. As is seen in Figure 2-6, the error in the cross-validation and test set decreases as the training set size increases. As such, adding training examples may improve the performance of a machine learning algorithm fit that suffers from high variance [21].

Looking at Figure 2-6 there are some important things to point out. Note that when the training set size is small, the training error is always very small and the test and cross validation error is always very high. This is because it is very easy for a machine learning algorithm to fit parameters to a small dataset. These parameters will fit the training set well but will struggle to generalize to new examples. Also of note is that the desired error of the algorithm is typically higher than the error on the training set. One expects the algorithm to learn the training data very well and then generalize slightly less well to new data. It would not make sense for an algorithm
to perform better on new examples than it did with the training set (assuming that, once learned, the $\theta$ parameters are fixed and not constantly being tweaked).

The choice of regularization parameter $\lambda$ also works to mitigate both high bias and high variance. Figure 2-7 below shows how the choice of $\lambda$ effects the various error measures.

![Figure 2-7: Cross Validation and Training Errors as a function of $\lambda$ in Ref. [24]](image)

As previously discussed, a small value of $\lambda$ can cause high variance and a large value of $\lambda$ can cause high bias. With this in mind, one realizes that an optimal value for $\lambda$ can be found that may improve the algorithm’s ability to generalize to new examples. The following are steps to follow to find an optimal value for $\lambda$

- Make a list of $\lambda$’s (i.e., 0, 0.01, 0.02, ..., 10, ...)
- Iterate through the $\lambda$ list and learn some $\theta$ parameters for the model(s)
- Calculate $J_{cv}$ (without the regularization term) for each of these models ($\theta$ parameters learned with the regularization term)
Select the model that has the lowest $J_{cv}$

Calculate $J_{cv}$ with this model to see if it generalizes well to new example.

In general, decreasing $\lambda$ works to mitigate high bias and increasing $\lambda$ works to mitigate high variance. It can also be shown that a smaller set of features can fix high variance and that adding additional features and/or adding polynomial features can fix high bias. When implanting any machine learning algorithm, it is important to keep these steps in mind when working towards finding the optimal middle ground between fitting the training set well and generalizing to new examples well [21].

Another important supervised machine learning algorithm to discuss is the Support Vector Machine (SVM). SVM is similar to both logistic regression and neural networks but is often times a “cleaner and more powerful” [21] algorithm. Just as with the previously discussed algorithm, there is a cost function that varies with the $\theta$ parameters that must be minimized in order to find the optimal $\theta$ parameters that fit the dataset. The SVM cost function has a similar form as the logistic regression cost function but with the slight modification that it uses the hinge loss function [20] instead of the sigmoid function. The regularized logistic regression cost function, as previously discussed is:

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$  \hspace{1cm} (20)

And inserting the definition of the hypothesis this equation becomes
Recall that this cost function is made up of two terms that correspond to two different cost functions depending on whether \( y = 1 \) or \( y = 0 \) (reference Figure 9). Noting that \( z = \theta^T x \) this can be written as:

\[
\text{cost}_0(z) = -\log \left( 1 - \frac{1}{1 + e^{-z}} \right) \\
\text{cost}_1(z) = -\log \left( \frac{1}{1 + e^{-z}} \right)
\]

The subscripts on the cost functions denote whether \( y = 1 \) or \( y = 0 \). With an SVM, the key idea is to modify the first and second terms of the logistic regression cost function such that when \( \theta^T x > 1 \), \( \text{cost}_1(z) = 0 \) and when \( \theta^T x < -1 \), \( \text{cost}_0(z) = 0 \) [21]. Figure 2-8 below shows the hinge loss function and the sigmoid function plotted together.
Note that when the cost function does not equal zero, the hinge loss function uses a straight increasing or decreasing line to define the cost, rather than the sigmoid curve. As such, the two components of the SVM cost function can be written as:

\[
\begin{align*}
    cost_0(z) &= \max(0, k(1 + z)) \\
    cost_1(z) &= \max(0, k(1 - z))
\end{align*}
\]

where “k is an arbitrary constant defining the magnitude of the slope of the line” [21].

Combining these two terms into one cost function and noting that the SVM convention is to use a regularizing factor \( C = \frac{1}{\lambda} \), the SVM cost function is expressed as:

\[
J(\theta) = C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2
\]

This cost function must be minimized in order to find the optimal \( \theta \) parameters that fit the dataset. Up to this point, the only difference between SVM and Logistic Regression is the
slightly different form of cost function. However, what differentiates SVM over the previously discussed algorithms is its use of kernel functions to build its decision boundaries [21].

SVM is known as a large margin classifier because its decision boundary is “as far away as possible” from the examples in each output class [21]. To achieve this, SVM employs kernel functions in order to fit \( \theta \) parameters to complex, non-linear functions that aim to create the large margin. The purpose of a kernel function is to take all the input features (variables), \( x \), and map them to new features based on their proximity to landmarks. A simple and common choice for these landmarks are the training examples themselves, which will be discussed further. There are many different types of kernel functions but the two most commonly used are the Linear Kernel and the Gaussian Kernel. The Linear Kernel is actually not a kernel at all, it is simply using the input features in the above cost function. The Gaussian Kernel employs the similarity function to define new training features, \( f_i \), and is mathematically expressed as follows:

\[
f_i = \text{similarity}(x, l^{(i)}) = \exp \left( -\frac{\|x - l^{(i)}\|^2}{2\sigma^2} \right) = \exp \left( -\frac{\sum_{j=1}^{m}(x_j - l^{(i)})^2}{2\sigma^2} \right)
\]

(27)

where \( \|x - l^{(i)}\| \) is the Euclidean distance between each training example, \( x \), and a landmark, \( l^{(i)} \), and \( \sigma^2 \) is the bandwidth parameter that controls how smoothly \( f_i \) varies when the distance between the training example and the landmark changes. The equation for \( f_i \) has two tunable values: \( \sigma^2 \) and \( l \). When \( \sigma^2 < 1 \) the value of \( f_i \) falls to 0 more quickly. On the other hand, when \( \sigma^2 > 1 \) the value of \( f_i \) falls to 0 more slowly and smoothly. As such, this term can be thought of as a weighting factor that favors/penalizes input variables based on their proximity to the landmarks. As previously mentioned, the standard choice for the landmarks are the locations of the input variables themselves. This is mathematically expressed as:
\[ l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)} \]  

(28)

It follows then that for each training example, \( x^{(i)} \), the goal is to calculate the value of its similarity function with respect to each landmark \{\( l^{(1)}, l^{(2)}, \ldots, l^{(m)} \)\}. This endeavor yields a vector of similarity values, \( F^{(i)} \), such that:

\[
F^{(i)} = \begin{bmatrix}
    f_1^{(i)} \\
    \vdots \\
    f_m^{(i)}
\end{bmatrix}
\]  

(29)

where

\[
\begin{align*}
    f_1^{(i)} &= \text{similarity}(x^{(i)}, l^{(1)}),
    f_2^{(i)} &= \text{similarity}(x^{(i)}, l^{(2)}), \ldots, \\
    f_m^{(i)} &= \text{similarity}(x^{(i)}, l^{(m)})
\end{align*}
\]  

(30)

Similar to the convention used in linear and logistic regression of including \( x_0 = 1 \), \( f_0^{(i)} = 1 \) is typically included in the \( F^{(i)} \) vector. Once \( F^{(i)} \) is computed for each input variable, the set of \( F^{(i)} \)'s becomes the input features with which the SVM cost function is evaluated with. To that end, the training objective for an SVM is mathematically expressed as:

\[
\min_{\theta} = C \sum_{i=1}^{m} \gamma^{(i)} \text{cost}_1(\theta^T F^{(i)}) + (1 - \gamma^{(i)}) \text{cost}_0(\theta^T F^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2
\]  

(31)

Just as with the previously discussed machine learning algorithms, it is useful to discuss how to deal with a fit that suffers from either high bias or high variance. Recall that \( C = 1/\lambda \). With that in mind, larger values of \( C \) (small \( \lambda \)) are more prone to high variance (overfitting) and smaller values of \( C \) (large \( \lambda \)) are more prone to high bias (underfitting). Furthermore, larger values of \( \sigma^2 \) in the Gaussian Kernel function are more prone to higher bias while smaller values of \( \sigma^2 \) are
more prone to higher variance. Again, as with the previously discussed machine learning
algorithms, tuning these parameters involves testing different values of them and choosing the
model that has the lowest error on the cross-validation set, $J_{cv}(\theta)$. The SVM algorithm can also
be applied to multiclass classification by employing either the one vs. all or one vs. one method
that was previously discussed in the logistic regression section.

Another topic to discuss in regards to SVMs is how to decide which kernel function to
use. As mentioned earlier, the most popular kernel choices are the Linear Kernel and the
Gaussian Kernel [21]. The Linear Kernel simply passes the unchanged input variables to the
cost function. The Linear Kernel should be used when $n$, the number of input variables is large
and/or when $m$, the number of training examples is small. This is intuitive because when the
number of input variables is large, it becomes computationally expensive to employ the Gaussian
Kernel function. The Gaussian Kernel is recommended when $n$ is small and/or $m$ is large. It is
also important to note that feature scaling should be performed when using the Gaussian Kernel
function. Given features that have varying magnitudes, the Gaussian Kernel function would get
dominated by features with large values because it squares the distance between each feature and
landmark. It is in the researcher’s best interest that the SVM pay equal attention to all of the
input features and feature scaling should be implemented to ensure this. Another option for the
kernel function is the Polynomial Kernel which takes the form

$$k(x, l) = (x^T l + a)^b$$  (31)

where $a$ and $b$ are constants. This is not a common choice for the kernel function because it
almost always performs worse than the Gaussian Kernel function[21]. Other choices for the
kernel function include the String Kernel, the Chi-Square Kernel, and the Histogram Intersection
Kernel [21]. Again, when deciding between kernel functions to use, it is best practice to make multiple models, train them, and choose the model with lowest $J_{cv}(\theta)$.

Now that a basic understanding of commonly used machine learning algorithms has been established, the next logical question ask is how to choose which algorithm is best for a given supervised classification problem. A general guideline can be established purely from the standpoint of the number of input variables, $n$, and the number of training examples, $m$ [21]. If $n$ is large relative to $m$ (i.e., $n = 10,000 +$ and $m = 10 - \sim 1,000$), it is best to use logistic regression or an SVM with a Linear Kernel. If $n$ is small and $m$ is large (i.e., $n = 1 - 1000$ and $m = 50,000 +$), it is again best to try and create and/or add more input variables and also use logistic regression or an SVM with a Linear Kernel. Logistic Regression and SVMs with Linear Kernels tend to give similar results, so the choice comes down to computational cost as well as which model has the lowest $J_{cv}(\theta)$. If $n$ is small and $m$ is intermediate (i.e., $n = 1 - 1000$ and $m = 10 - \sim 10,000$), it is best to use an SVM with a Gaussian Kernel. The above should be used as a guideline to narrow down which machine learning algorithms may be worth it to try on a given problem. Once narrowed down, though, it is best to build multiple models, train them, and choose the model which has the lowest $J_{cv}(\theta)$. This thesis does not attempt to be design or modify a machine learning algorithm as Matlab’s Classification Learner toolbox provides a GUI in which all of the discussed algorithms have been implemented and are available for use. As such, these guidelines were used to identify that an SVM with either a Linear or Gaussian Kernel will be suitable to test the hypothesis. The rest of this chapter focuses on the dataset used with the above choice of machine learning algorithm. The dataset’s background, what other researchers have done with it in the past, and this thesis’ novel contribution is discussed.
2.2 The C-MAPSS Dataset

Up to now, the discussion has been focused on how to implement machine learning algorithms on a dataset. There has also been discussion on the scarcity of useful, robust datasets with run-to-failure data in the gas turbine world. NASA’s C-MAPSS software “simulates an engine model of the 90,000 lb thrust class” [30]. The software was built in MATLAB® and Simulink® and consists of various input parameters, controllers, and operational conditions that the user can specify in order to create their desired simulation. C-MAPSS allows the user to simulate degradation in any of the five major rotating components of the engine; the fan, LPC, HPC, HPT, and LPT. A flowchart showing the layout of the engine components simulated in C-MAPSS is shown in Figure 2-9 below.

![Figure 2-9: Schematic and Flowchart of Gas Turbine Engine Modeled in C-MAPSS Software in Ref. [30]](image)

All of these components are present in the majority of gas turbine engines, which makes C-MAPSS a valuable tool for running realistic, high fidelity engine simulations. [30] goes into great detail about how C-MAPSS was used to create the dataset used in this thesis. To summarize, response surfaces were built in [9] Cycle Deck, a “thermo-dynamical simulation model” for gas turbine engines. These response surfaces show how the sensor outputs of the
simulation vary when the input parameters of flow and efficiency are changed. Examples of these response surfaces, adapted from [9], are shown below in Figure 2-10.

Using these response surfaces, [9] operational margins are defined based on engine component performance maps within Cycle Deck. These operational margins are used to define the health
An example of the operational margin on the HPC, also adapted from [9], is shown below in Figure 2-11.

![Figure 2-11: Operational Margin of HPC Fault in Cycle Deck in Ref. [9]](image)

The health index goes to 0 once any operational margin is hit, which represents failure. Using these response surfaces and operational margins, a complete run-to-failure dataset of sensed outputs can be generated using a software package such as C-MAPSS, which was done in [30] by Saxena et al. In order to simulate run-to-failure, Saxena et al. imposed an exponential rate of change to the input parameters of flow and efficiency, as well as choosing a random initial deterioration no greater than 1% of the healthy initial condition as defined in [31]. This process is mathematically is described as follows. Saxena et al. states that “common to all degradation models is the exponential behavior of the fault evolution” [30]. As such, they propose a generalized equation for wear

\[ w = Ae^{B(t)} \]  

(32)
which is further manipulated by “assuming an upper wear threshold, $t_{hw}$, that denotes an operational limit beyond which the component/subsystem cannot be used” [30]. Wear is then subtracted from the upper wear threshold and normalized with respect to the upper wear threshold in order to define the health index as a function of time

$$h(t) = 1 - \frac{A e^{B(t)}}{t_{hw}}$$

(33)

To make the equation easier to work with, Saxena et. al. defines

$$\frac{A}{t_{hw}} = e^a$$

(34)

$$B(t) = t^b$$

(35)

which yields

$$h(t) = 1 - \exp\{at^b\}$$

(36)

where $a$ and $b$ are parameters associated with the component/subsystem whose health index is being calculated. The initial degradation, $d$, is incorporated as an additive term such that

$$h(t) = 1 - d - \exp\{at^b\}$$

(37)

This study explores degradation of rotating engine components described by the efficiency of the component and the flow through the component. As such, health indices for the flow and efficiency of an engine component are defined as follows

$$e(t) = 1 - d_e - \exp\{a_e(t)t^{b_e(t)}\}$$

(38)
\[ f(t) = 1 - d_f - \exp\{a_f(t)t^{b_f(t)}\} \]  

(39)  

Since both of these parameters define the operability margin for a given component, the overall health index is defined as the aggregate of the flow and efficiency health indices.

\[ H(t) = g(e(t), f(t)) \]  

(40)  

where

\[ g(e(t), f(t)) = \min(m_{fan}, m_{HPC}, m_{HPT}, m_{EGT}) \]  

(41)  

where \( m_x \) is the current position relative to the operability margin for a given rotating component [30].

Saxena et. al. describes their modeling process as follows [30]:

- Randomly choose initial deterioration
  - \( e_0 \in \{0.99, 1\} \)
  - \( f_0 \in \{0.99, 1\} \)
- “Impose exponential rate of change for flow and efficiency loss…the direction and evolution of faults is constrained by”
  - \( f_i, e_i \leq 1\% \)
  - \( a_k \in [0.001, 0.003] \)
  - \( b_k \in [1.4, 1.6] \), \( k = 1, 2 \)
- Stop simulation when \( H = 0 \)
- “Superimpose measurement noise to the output data”
To summarize, flights are simulated and the exponential change of flow and efficiency loss runs its course until the first operational margin of the engine is hit. This could be any of the operational margins, but as soon as the first one is reached, the health index reaches 0 and the engine is considered to have failed. A key assumption in producing this dataset is that there is no sudden degradation of any component during a flight. This allows one data point per flight to characterize the health of the engine at that point in time. Furthermore, to simulate between-flight maintenance, Saxena et. al. allowed the flow and efficiency to improve within allowable limits so that these parameters are not monotonically decreasing over the duration of the simulation. Since this study simulates sensor outputs, it is also important to simulate the noise associated with sensor measurements. To simulate non-trivial noise, the researchers added a mixture noise distribution, made up of multiple “simplistic normal noise distributions” to the dataset [30]. Once the simulation was completed and the dataset was gathered, the sensor outputs of the C-MAPSS model were mapped to these response surfaces from [9] to ensure that the sensor data makes physical sense. The software outputs sensor measurements corresponding to various temperatures, pressures, flow rates, and other system parameters. The input parameters to the C-MAPSS simulation are shown below in Table 2-1.
The researchers used the flow and efficiency modifiers above to simulate degradation in any of the rotating components, which in turn produced a run-to-failure dataset of sensed outputs. The sensed outputs are shown below in Table 2-2.

*Table 2-2: Sensed Outputs from C-MAPSS Engine Simulation in Ref. [30]*

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>Total temperature at fan inlet</td>
<td>°R</td>
</tr>
<tr>
<td>T24</td>
<td>Total temperature at LPC outlet</td>
<td>°R</td>
</tr>
<tr>
<td>T30</td>
<td>Total temperature at HPC outlet</td>
<td>°R</td>
</tr>
<tr>
<td>T50</td>
<td>Total temperature at LPT outlet</td>
<td>°R</td>
</tr>
<tr>
<td>P2</td>
<td>Pressure at fan inlet</td>
<td>psia</td>
</tr>
<tr>
<td>P15</td>
<td>Total pressure in bypass-duct</td>
<td>psia</td>
</tr>
<tr>
<td>P30</td>
<td>Total pressure at HPC outlet</td>
<td>psia</td>
</tr>
<tr>
<td>Nf</td>
<td>Physical fan speed</td>
<td>rpm</td>
</tr>
<tr>
<td>Nc</td>
<td>Physical core speed</td>
<td>rpm</td>
</tr>
<tr>
<td>epr</td>
<td>Engine pressure ratio (P50/P2)</td>
<td>--</td>
</tr>
<tr>
<td>P30</td>
<td>Static pressure at HPC outlet</td>
<td>psia</td>
</tr>
<tr>
<td>phi</td>
<td>Ratio of fuel flow to Ps30</td>
<td>pps/psi</td>
</tr>
<tr>
<td>NfR</td>
<td>Corrected fan speed</td>
<td>rpm</td>
</tr>
<tr>
<td>NfRe</td>
<td>Corrected core speed</td>
<td>rpm</td>
</tr>
<tr>
<td>BPR</td>
<td>Bypass Ratio</td>
<td>--</td>
</tr>
<tr>
<td>farB</td>
<td>Burner fuel-air ratio</td>
<td>--</td>
</tr>
<tr>
<td>hBleed</td>
<td>Bleed Enthalpy</td>
<td>--</td>
</tr>
<tr>
<td>Nf_dmd</td>
<td>Demanded fan speed</td>
<td>rpm</td>
</tr>
<tr>
<td>PCfR_dmd</td>
<td>Demanded corrected fan speed</td>
<td>rpm</td>
</tr>
<tr>
<td>W31</td>
<td>HPT coolant bleed</td>
<td>lbm/s</td>
</tr>
<tr>
<td>W32</td>
<td>LPT coolant bleed</td>
<td>lbm/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters for calculating the Health Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>T48 (EGT)</td>
</tr>
<tr>
<td>SmFan</td>
</tr>
<tr>
<td>SmLPC</td>
</tr>
<tr>
<td>SmHPC</td>
</tr>
</tbody>
</table>
After the entire process was completed, the dataset was made available for the Prognostics and Health Management (PHM) data competition at PHM’08. Since then, many researchers have utilized the C-MAPSS datasets to develop and improve prognostic algorithms to make predictions on the health of gas turbine engines. Ramasso et. al. propose the following characteristics that make the C-MAPSS dataset “very useful and suitable for developing prognostic algorithms” [14].

1. “Data represent a multi-dimensional response from a complex non-linear system from a high-fidelity simulation that very closely models a real system
2. These simulations incorporated high levels of noise introduced at various stages to accommodate the nature of variability generally encountered
3. The effects of faults are masked due to operational conditions, which is yet another common trait of most operational systems
4. Data from plenty of units is provided to allow algorithms to extract trends and build associations for learning system behavior.” [14]

The C-MAPSS .zip file from the NASA website [28] comes with four datasets of varying complexity, with each exhibiting a different number of fault modes and operating conditions. Each training set contains run-to-failure data across multiple engines and each test set contains engine data with an unknown remaining useful life (cycles before failure). There are also text files that give the final remaining useful life for each engine in the test set. Utilizing both a training set and its corresponding test set, researchers can train a machine learning algorithm(s) using the training set and test its ability to generalize to new data on the test set. Table 2-3 below shows the makeup of each of the datasets that come in the C-MAPSS .zip file.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Fault Modes</th>
<th>#Conditions</th>
<th>#Train Units</th>
<th>#Test Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbofan data from NASA repository</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>#2</td>
<td>1</td>
<td>6</td>
<td>260</td>
<td>259</td>
</tr>
<tr>
<td>#3</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>#4</td>
<td>2</td>
<td>6</td>
<td>249</td>
<td>248</td>
</tr>
</tbody>
</table>
Since each dataset has a different level of complexity, it is difficult to use all of them for a single study and/or compare results amongst all of the datasets [14]. As such, dataset #1, the simplest and least complex dataset, is used for this thesis. This allows the novel physics-based filtering technique to be applied to the entire dataset without having to consider multiple operating conditions and fault modes. The filter, though, can surely be applied to these datasets and suggestions for future work involving the other datasets is provided at the end of this thesis.

2.3 Literature Review- Previous Studies Using C-MAPSS Dataset
Ramasso et. al. provides a comprehensive literature review of all the published research papers that utilize the C-MAPSS datasets [14]. The researchers scanned the literature to find over 70 publications that cited the C-MAPSS dataset. The list was then reduced to 40 unique journal papers by filtering out similar papers by the same authors and any articles that did not directly use the any of the datasets. The paper goes on to summarize the methodology and findings of the journal papers. Since that paper is already written, there is no need to regurgitate its findings. As such, summary figures and tables adapted from Ramasso et. al. and a brief discussion are presented in this section [46-80]. Figure 2-12 below shows the breakdown of which datasets were used.

![Figure 2-12: Breakdown of C-MAPSS Datasets Usage in Ref. [14]](image-url)
There are over 40 papers shown in the graph on the left, but some of those papers may have been similar efforts by the same authors and thus those were filtered out of the final 40. Regardless, most published papers utilizing the C-MAPSS datasets use data that is currently available in the NASA repository. Both the data from the data challenge and the simulator itself are not currently available. Of the 31 papers that used the data in the NASA repository, 70% of those used dataset #1 which is the dataset used in this thesis. Further, only 11 out of the 40 unique journal papers utilized a full training/testing. This thesis uses the full training and testing set #1. Of the 40 journal papers, some used classification (referred to as detection), some used regression (referred to as prediction), and some used classification as a processing step towards a larger regression problem. Table 2-4 below shows the breakdown of these choices.

**Table 2-4: Breakdown of Problem Framing Using C-MAPSS Datasets in Ref. [14]**

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Publication ID</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>6, 10, 11, 12, 13, 31, 33, 34, 35, 37</td>
<td>10/40</td>
</tr>
<tr>
<td>Prediction</td>
<td>1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 16, 17, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 32, 34, 36, 37, 38, 39, 40</td>
<td>31/40</td>
</tr>
<tr>
<td>Other</td>
<td>15, 18, 26</td>
<td>3/40</td>
</tr>
</tbody>
</table>

The papers that correspond to each publication ID are found in the appendix of Ramasso et. al. Table 2-5 below shows what classification methods were used for papers that used any classification at any point.

**Table 2-5: Breakdown of Classification Methods Using C-MAPSS Datasets in Ref. [14]**

<table>
<thead>
<tr>
<th>Classification method</th>
<th>Publication ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>6, 10, 11, 12, 13, 26, 31, 35</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>1, 7, 11, 20, 24, 32</td>
</tr>
<tr>
<td>Partially supervised</td>
<td>27, 33</td>
</tr>
</tbody>
</table>

48
Between Tables 2-4 and 2-5, there are many valid ways to frame research problems using these datasets. This thesis poses a supervised detection (classification) problem, with the focus being on how faulty data effects the machine learning models’ ability to classify examples. Table 2-6 below summarizes which machine learning methods were used in some of the publications.

*Table 2-6: Breakdown of Machine Learning Methods Used on C-MAPSS Datasets in Ref. [14]*

<table>
<thead>
<tr>
<th>Methods</th>
<th>Publication ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadratic fit, Bayesian updating</td>
<td>4</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>5</td>
</tr>
<tr>
<td>Kernel regression, RVM</td>
<td>7</td>
</tr>
<tr>
<td>RVM</td>
<td>16</td>
</tr>
<tr>
<td>Gamma process</td>
<td>17</td>
</tr>
<tr>
<td>Linear, Bayesian updating</td>
<td>19</td>
</tr>
<tr>
<td>RVM, SVM, RNN, Exponential and quadratic fit, Bayesian updating</td>
<td>21</td>
</tr>
<tr>
<td>Exponential fit</td>
<td>28</td>
</tr>
<tr>
<td>Wiener process</td>
<td>29</td>
</tr>
<tr>
<td>Copula</td>
<td>30</td>
</tr>
<tr>
<td>HMM, LS SVR</td>
<td>34</td>
</tr>
</tbody>
</table>

Logistic regression and Support Vector Machines, both of which have been discussed previously, have been applied to the C-MAPSS datasets. As previously mentioned, the C-MAPSS datasets are well-suited to use to build prognostic models in various ways. The datasets were released after the PHM’08 challenge ended so that researchers can try their hand at building models and can compare results. Tables 2-4, 2-5, and 2-6 show the many different possible ways to frame problems on these datasets, and these tables are far from exhaustive. What makes this thesis stand out compared to these 40 papers is that it frames the problem in a unique way (supervised classification using SVM models) and explores the effects of faulty data within the C-MAPSS dataset #1 and proposes a way to mitigate these effects, which has not been done before.
2.4 Novel Contributions of This Thesis

Two physics principles were applied to the dataset in order to filter out any data that violates laws of physics. Data like this can be encountered due to many reasons including faulty sensors, misreading of sensors and sensor noise. Applying fundamental physics principles to a dataset is a critical assessment to ensure that the data makes sense and can be trusted.

2.4.1 Second law of thermodynamics applied to the compressor

The first physical principle is the second law of thermodynamics applied to the compressor, with a control volume that starts at the fan inlet and ending at the HPC outlet [26]. Referring to the flowchart in Figure 2-9, the fan is the only rotating component before the compressor inlet. Further, Table 2-2 shows that the total temperature at the fan inlet is given. The rest of the temperatures given in Table 2-2 are all outlet temperatures. To apply the second law of thermodynamics to the compressor, the inlet total temperature is needed. By using a control volume that starts at the fan inlet and ends at the HPC outlet, the total temperature at the LPC inlet is assumed to be the total temperature at the fan inlet. This makes sense because there is no engine component between the fan and LPC that causes the total temperature at the fan inlet to change before the air enters the compressor. A diagram of the compressor and its control volume is shown below in Figure 2-13 below.
Figure 2-13: Diagram of Compressor Showing the Inlet, Outlet, Control Volume, and Work Input in Ref. [27]

In this situation, the inlet to the compressor is the inlet of the LPC and the outlet is the outlet of the HPC, and the multi-stage compressor section is modeled as one compressor. This approach is applicable as long as there is no additional processes (e.g. intercooling) between the compressor stages. Air flows into the inlet at a certain total temperature and total pressure given in the dataset and then flows out at a higher total temperature and pressure also given in the dataset.

The first law of thermodynamics is

$$\delta q - \delta w = du$$  \hspace{1cm} (42)

By incorporating the second law of thermodynamics, this can be rewritten as

$$Td\delta s - Pd\delta v = du$$  \hspace{1cm} (43)

Enthalpy is defined as

$$h = u + P\nu$$  \hspace{1cm} (44)

The differential of enthalpy becomes
\[ dh = du + pdv + vdP \]  
\hfill (45)  

Combining the equations 43 and 45 yields

\[ Tds + vdp = dh \]  
\hfill (46)  

Then dividing by \( T \), setting \( ds = 0 \) (for an ideal compressor, entropy generation will be zero), defining \( dh = c_p dT \), and applying the ideal gas law \( v = RT/P \) yields

\[ 0 = c_p \frac{dT}{T} - R \frac{dP}{P} \]  
\hfill (47)  

Then

\[ c_p \frac{dT}{T} = R \frac{dP}{P} \]  
\hfill (48)  

If the specific heats can be assumed constant, integrating both sides yields

\[ \ln \left( \frac{T_2}{T_1} \right)^{c_p} = \ln \left( \frac{P_2}{P_1} \right)^R \]  
\hfill (49)  

By definition, the specific gas constant, \( R = c_p - c_v \) and the specific heat ratio, \( \gamma = c_p/c_v \).

Rearranging yields

\[ \frac{T_2}{T_1} = \left( \frac{P_2}{P_1} \right)^{\frac{\gamma - 1}{\gamma}} \]  
\hfill (50)  

It is important to note that this equation only applies for isentropic processes, so equation 50 should be written as
\[
\frac{T_{2s}}{T_1} = \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}} \tag{51}
\]

The isentropic outlet temperature, \(T_{2s}\), should always be less than the actual outlet temperature \(T_2\).

Applying this to a gas turbine, it can be shown that, for a compressor:

\[
\text{CPR} = \frac{P_{t,HPC,\text{out}}}{P_{t,LPC,\text{in}}} = \left(\frac{T_{tS,HPC,\text{out}}}{T_{tLPC,\text{in}}}\right)^{\frac{\gamma}{\gamma-1}} \tag{52}
\]

It should be noted that equation 52 uses total notation while the rest of the derivation uses static notation. Equation 52 is valid for both static and total properties [26]. Again, this equation assumes an isentropic compressor, which is not true in practice. As such, this equation can be used to calculate the isentropic HPC outlet temperature, \(T_{tS,HPC,\text{out}}\), which can then be compared to the \(T_{t,HPC,\text{out}}\) sensor measurement in the dataset. If \(T_{t,HPC,\text{out}} \geq T_{tS,HPC,\text{out}}\) then the data makes physical sense. If this does not hold true for any cycle, there is an issue with the measurements and all sensor values for the given cycle should be deleted. The adiabatic efficiency of the entire compressor section can also be calculated using compressor thermodynamics [26]. The work input per unit mass into the compressor, assuming constant specific heat, is calculated as

\[
W_c = c_p(T_{t,HPC,\text{out}} - T_{t,LPC,\text{in}}) = \frac{c_p T_{tLPC,\text{in}}}{\eta_c} \left(\text{CPR} \frac{\gamma-1}{\gamma} - 1\right) \tag{53}
\]

Equation 53 can be rearranged to solve for the compressor’s adiabatic efficiency as follows.
\[ \eta_c = \frac{T_{t,LP,C,in}}{(T_{t,HPC,out} - T_{t,LP,C,in})} \left( \frac{\gamma - 1}{\gamma} \right) \]  

(54)

By definition, the adiabatic efficiency should always be equal or less than unity. If this is not the case, all sensor values for the given cycle should be deleted.

2.4.2 Conservation of Energy: Combustor
The second concept of interest is using conservation of energy at the combustor section to determine the inlet temperature of the HPT. With this calculated value, turbine thermodynamics can be applied to the entire turbine section of the engine. The heat exchanged in the combustor section is calculated as

\[ Q = \dot{m}_f (\Delta h_{comb}) \]  

(55)

where \( \Delta h_{comb} \) is the LHV of the jet fuel, which is assumed to be Jet-A [33]. The energy balance across the burner is mathematically expressed as follows

\[ \dot{Q} = (\dot{m}_{air} + \dot{m}_{fuel})c_p T_{t,HPT,in} \]

\[ - (\dot{m}_{air} c_{p,air} T_{t,HPC,out} + \dot{m}_{fuel} c_{p,fuel} T_{t,HPC,out}) \]  

(56)

noting that the burner fuel-air-ratio is \( \phi = \frac{\dot{m}_{fuel}}{\dot{m}_{air}} \), that \( \dot{m}_{air} + \dot{m}_{fuel} = \dot{m}_{air} (1 + \phi) \), and assuming that \( c_{p,air} = c_{p,fuel} = c_p \), the energy balance is rewritten as

\[ \dot{m}_{fuel}(\phi \Delta h_{comb}) = \dot{m}_{air} (1 + \phi) c_p (T_{t,HPT,in} - T_{t,HPC,out}) \]  

(57)

Using this equation, \( T_{t,HPT,in} \) is solved for and turbine thermodynamics can now be applied to the entire turbine section of the engine. This step is required because \( T_{t,HPT,in} \) is not a directly
sensed measurement in the dataset. A diagram of the combustor and the turbine control volume is shown below in Figure 2-14.

![Diagram of Combustor and Turbine](image)

_Figure 2-14: Diagram of Combustor and Turbine (left) and Turbine Showing the Inlet, Outlet, Control Volume, and Work Output (right) in Refs. [81][27]_

In this situation, the inlet to the turbine is the inlet of the HPT and the outlet is the outlet of the LPT. Similar to the compressor, this allows the turbine section to be modeled as one single turbine. The air-fuel mixture leaves the burner and flows into the turbine inlet at a certain total temperature and pressure. The mixture’s energy is then converted into a work output that is used as thrust for the engine and the air-fuel mixture flows out of the outlet.

Assuming constant pressure combustion in the burner section, the isentropic relation between inlet and exit temperature and pressure is written as
\[ \frac{TPR}{TPR_HPT_{in}} = \left( \frac{T_{t_{LS},LPT_{out}}}{T_{t_{HPT}_{in}}} \right)^{\frac{\gamma}{\gamma - 1}} \] (58)

This equation assumes an isentropic turbine, which is not true in practice. As such, this equation can be used to calculate the isentropic turbine outlet temperature, \( T_{t_{LS},LPT_{out}} \), which can then be compared to the \( T_{t_{LPT},out} \) sensor measurement in the dataset. If \( T_{t_{LPT},out} \geq T_{t_{LS},LPT_{out}} \) then the data makes physical sense. If this does not hold true for a particular cycle, all sensor values for the given cycle should be deleted. The adiabatic efficiency of the entire turbine section can also be calculated using turbine thermodynamics. The work output per unit mass into the turbine, assuming constant specific heat, is calculated as

\[ \dot{W}_t = c_p (T_{t_{HPT}_{in}} - T_{t_{LPT},out}) = \eta_t c_p T_{t_{HPT}_{in}} \left( 1 - TPR \frac{\gamma - 1}{\gamma} \right) \] (59)

Equation 59 can be rearranged to solve for the turbine’s adiabatic efficiency as follows

\[ \eta_t = \frac{T_{t_{HPT}_{in}} - T_{t_{LPT},out}}{T_{t_{HPT}_{in}} \left( 1 - TPR \frac{\gamma - 1}{\gamma} \right)} \] (60)

By definition, the adiabatic efficiency should always be equal to or less than unity. If this is not the case, all sensor values for the given cycle should be deleted.

With these physics concepts in mind, the physics-based filter is implemented. In order to test its usefulness, a test matrix is devised to explore scenarios in which varying degrees of bad data are added to the testing dataset. Two scenarios that would mirror the effects of real-life situations were focused on. First, the scenario where an uncalibrated sensor is used for data collection was explored. In this scenario, the sensor measurements on the engine that the uncalibrated sensor is wired to may be off from their real value by a constant factor. The effects
varying this constant factor were also studied. For this scenario, HPC and LPT outlet temperatures are multiplied by two constant factors; one that makes the measurements harshly violate physics, and another that makes the measurements mildly violate the physics. Engines 2, 6, 10, 14, and so on up to 100 had their HPC outlet temperature made faulty, while engines 4, 8, 12, 16, and so on up to 98 had their LPT outlet temperature made faulty; the odd numbered engines had their data left untouched. In the harsh scenario the HPC outlet temperature is multiplied by 0.8 and the LPC outlet temperature is multiplied by 0.5, which brings the adiabatic efficiency of both the compressor and turbine section up to about 1.25. In the mild scenario, the HPC outlet temperature is multiplied by 0.9 and the LPC outlet temperature is multiplied by 0.8, which brings the adiabatic efficiency of both the compressor and turbine section up to about 1.05.

The second scenario that is explored is when a component along the path of data transmission from the sensor to the DAQ computer is faulty and/or nearing the end of its life. In this scenario, the sensor readings may intermittently be off their real value by a constant factor. The effects of varying this constant factor were also studied. Both the HPC and LPT outlet temperatures are multiplied by two constant factors, using the same approach as the first scenario; one that makes the measurements harshly violate physics and one that makes the measurements mildly violate the physics. 1000 data points are randomly chosen from the test set to make faulty, and the same 1000 data points are faulty in each of the analyses. A random 500 of these 1000 points had faulty HPC outlet temperatures while the other 500 points had faulty LPC outlet temperatures. The same constants from the first scenario were used to represent the harsh and mild cases.
Preparing the Training Set and Training Machine Learning Models

Given a suitable dataset, extensive work must be done to make the data usable in building a prognostic model for gas turbines [14]. This section first discusses how the training set data is processed to train the machine learning models. Later on in this section discussion is presented on how the test set data is processed to incorporate bad data. The full C-MAPSS training set #1 contains run-to-failure data for 100 engines and a total of 20631.

The vast majority of real-life sensor data is muddied by noise; the raw data may not say anything useful about the state of the machine. As such, the first step in building a prognostic model is the signal processing step. This can involve smoothing the sensor signals to diminish the effect of noise, or converting the sensor values into a more useful measurement (i.e., calculating shaft speed in RPM from a tachometer voltage reading). Some techniques used for this step are the Kalman Filter [15], Gaussian Kernel Smoothing [16] and Principal Component Analysis [17]. In this thesis, the data is smoothed using a simple moving average filter using Matlab’s \texttt{movmean} function. This function works by replacing a given data point in a column by the moving average of the column calculated within a sliding window of a specified length. Specifically, this thesis uses a sliding window of length 5 looking backwards only. This means that a given data point is replaced by the mean calculated using that data point and the four previous data points. Since mixture distribution noise was superimposed onto the sensed outputs from the C-MAPSS simulation, a simple moving average filter implemented in this fashion should work to reduce artificial outliers in the dataset. This, in turn, allows predictive classifications to be more easily made on the dataset. In order to study the effectiveness of this filter, scenarios are tested where the SMA filter is not applied to the data.
The next step, which is closely related to signal processing, is the feature extraction step. This is where features, also known as variables, are selected for use in the prognostic model. PCA can also be used here as a way to reduce the dimensionality of the dataset. As previously discussed, too many features can cause the model to overfit the dataset, which means that the model will perform well on the training set but may not perform as well with a new testing dataset. The C-MAPSS dataset contains data from 21 sensors, seven of which are constant over the entirety of the dataset. These features are simply removed from the dataset as they would not improve the machine learning algorithm’s performance and would only serve to make the training and testing process more computationally expensive. The constant features are the total fan inlet temperature, total fan inlet pressure, engine pressure ratio, burner air fuel ratio, demanded fan speed, demanded corrected fan speed, and total pressure in the bypass duct. All other features previously shown in Table 2-2 are used in this study. This thesis also considers feature normalization affects the performance of the trained machine learning models. The magnitude of the remaining features in the training set range from 8-9000 so mean normalization makes the values of each feature much more similar. As such, some training sets consist of mean normalized features. To study the effectiveness of feature mean normalization, scenarios are also tested where feature mean normalization is not applied to the data. Between the SMA filter and mean normalization, there are now four cases of models that need to be trained: SMA filter with mean normalization, SMA filter without mean normalization, mean normalization without SMA filter, and neither SMA filter nor mean normalization.

In the spirit of run-to-failure data being scarce for gas turbine engines, the scenario of having less training data to work with is also explored. Given that the training set used contains data from 100 engines, it is interesting to evaluate the performance of machine learning
algorithms trained on only some of this data and compare it to the performance of machine learning algorithms trained on all of the data. To this end, the same machine learning algorithms are trained on a training set that contained run-to-failure data from only 10 of the engines. Ten engines are randomly selected from the training dataset and performed the exact same preparation on that set and the set with all engines. These ten random engines are units 15, 20, 47, 56, 64, 70, 71, 79, 80, and 91.

After signal processing and feature selection comes the health estimation and classification steps. This thesis chooses to combine the steps estimate the health of the engine by classifying training points based on their proximity to the engine’s last cycle before failure. The raw dataset is organized by cycle, meaning that the first data point for a given engine is cycle one, and the last data point is the numbered cycle that the engine failed at. Instead of using the given number of cycles, I chose to manipulate this column into time to failure (TTF). This is done by denoting the last cycle as when the TTF is 0, and the first cycle as when the TTF is equal to the total number of cycles remaining. For example, engine one in the training set went through 192 cycles before failing. In terms of TTF, the first cycle has a TTF of 191 and the last cycle has a TTF of 0. Two different classification schemes are used in order to train SVM algorithms on both multiclass classification and binary classification. The multiclass classification scheme is as follows:

- If the engine has more than 200 cycles remaining before failure (TTF greater than 200), it is classified as having a **long** time to maintenance
- If the engine has between 200 and 125 cycles remaining before failure (TTF between 200 and 126), it is classified as having a **medium** time to maintenance
- If the engine has between 125 and 50 cycles remaining before failure (TTF between 125 and 51), it is classified as having a **short** time to maintenance

- If the engine has 50 or less cycles remaining before failure (TTF of 50 or less), it is classified as having an **urgent** time to maintenance

Table 2-7 below shows the distribution of classifications amongst the training set using the above classification scheme.

**Table 2-7: Breakdown of Training Set Multiclass Classification**

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1958</td>
<td>9.49%</td>
</tr>
<tr>
<td>Medium</td>
<td>6073</td>
<td>29.44%</td>
</tr>
<tr>
<td>Short</td>
<td>7500</td>
<td>36.35%</td>
</tr>
<tr>
<td>Urgent</td>
<td>5100</td>
<td>24.72%</td>
</tr>
</tbody>
</table>

As previously discussed in the machine learning section of the introduction chapter, there are two ways to train a machine learning algorithm when it is asked to perform multiclass classification: the one vs. one method and the one vs. all method. Models are built and trained using both of these methods to evaluate which method yields better performance on both the training and testing sets. The binary classification scheme is as follows

- If the engine has more than 50 cycles remaining before failure, it is considered to be **healthy**

- If the engine has 50 or less cycles remaining before failure, it is considered to be **unhealthy**

Table 2-8 below shows the distribution of classifications amongst the training set using the above classification scheme.
Table 2-8: Breakdown of Training Set Binary Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>15531</td>
<td>75.28%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>5100</td>
<td>24.72%</td>
</tr>
</tbody>
</table>

Tables 2-9 and 2-10 below shows the distributions of classifications for multiclass and binary classifications amongst the training set with data from the 10 engines mentioned above.

Table 2-9: Breakdown of 10 Engine Training Set Multiclass Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>215</td>
<td>10.35%</td>
</tr>
<tr>
<td>Medium</td>
<td>602</td>
<td>28.98%</td>
</tr>
<tr>
<td>Short</td>
<td>750</td>
<td>36.11%</td>
</tr>
<tr>
<td>Urgent</td>
<td>510</td>
<td>24.55%</td>
</tr>
</tbody>
</table>

Table 2-10: Breakdown of Training Set Binary Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>1567</td>
<td>75.45%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>510</td>
<td>24.55%</td>
</tr>
</tbody>
</table>

Comparing the percentages in Tables 2-9 and 2-10 to those in Tables 2-7 and 2-8, it is clear that the all engine training sets and the 10 engine training sets have similar class distributions.

Incorporating multiclass and binary classification adds another dimension to the matrix of machine learning models trained on the training set. This matrix is shown below in Table 2-11.
<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>MultiClassification</th>
<th>EngineList</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>No</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>No</td>
</tr>
</tbody>
</table>
In total there are 48 trained models tested on a similar matrix of various test sets. The final step of preparing the training set is the prediction step. Each model is trained by importing the processed training set into Matlab’s Classification Learner application. From there, the desired algorithm is selected and trained on the given data. Once a model is trained, Classification Learner displays various data regarding the performance of the machine learning algorithm including scatter plots showing correct and incorrect classifications as well as
confusion matrices. Methods of evaluating the performance of the trained algorithm on both the training set and the testing set are discussed later on in this section. The following is a summary of the procedure used to prepare the training set.

- Import raw data into Matlab
- Apply SMA filter to the data (if necessary)
- Feature mean normalization (if necessary)
- Evaluate the TTF
- Health classification based on TTF and desired multiclass or binary classification
- Import into Classification Learner and train the models

After training all 48 models, each one of them is saved to evaluate their performance on the testing sets.

2.6 Preparing the Testing Sets and Testing the Trained Models
The C-MAPSS test set #1 contains data from 100 engines with a total of 13096 data points. The testing set does not contain complete run-to-failure data, but the number of cycles remaining after the last data point is provided in a text file. This allows the TTF, and thus the correct health classification, to be calculated for every data point so that the performance on the testing set can be evaluated. Tables 2-12 and 2-13 below show the distribution of classifications amongst the testing set using the multiclass and binary classification schemes.
Table 2-12: Breakdown of Testing Set Multiclass Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1959</td>
<td>14.9%</td>
</tr>
<tr>
<td>Medium</td>
<td>5993</td>
<td>45.76%</td>
</tr>
<tr>
<td>Short</td>
<td>4258</td>
<td>32.51%</td>
</tr>
<tr>
<td>Urgent</td>
<td>886</td>
<td>6.77%</td>
</tr>
</tbody>
</table>

Table 2-13: Breakdown of Testing Set Binary Classification

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>12210</td>
<td>93.23%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>886</td>
<td>6.77%</td>
</tr>
</tbody>
</table>

It is important to note that, in both classification scenarios, only 6.77% of the examples are classified as urgent/unhealthy. Engineers need to know when a gas turbine engine is close to failure so that they can plan accordingly. It would not be ideal for an engine component to fail while the engine is in use. As such, the urgent/unhealthy classification is the most important for the machine learning algorithm to make because it identifies engines that require immediate attention. This is an example of skewed data; there are only minimal examples of the classification that is most desirable to predict [21]. Discussion on how to evaluate algorithm performance in the presence of skewed classes is presented later in this chapter.

The only difference between the procedure to prepare the training set and the procedure to prepare the testing set is the incorporation of bad data into the test set. It is imperative that the bad data be incorporated prior to any data processing in order to mimic the real-life scenario of acquiring bad data that were previously discussed. The entire test matrix is too large to display.
in a table, but can be summarized in a numbered list. Each of the 48 trained models is tested with 9 variations of the test set, each with varying degrees of bad data within them.

1. Test set only contains good data
2. Test set contains 1000 random points of harshly faulty data
3. Physics-based filter applied to #2 to remove bad data points
4. Test set contains 1000 random points of mildly faulty data
5. Physics-based filter applied to #4 to remove bad data points
6. Test set contains harshly faulty data on the even units
7. Physics-based filter applied to #6 to remove bad data points
8. Test set contains mildly faulty data on the even units
9. Physics-based filter applied to #8 to remove bad data points

In each of these scenarios, the bad data is immediately added to the test set after it is imported into Matlab. After the bad data is incorporated, the preparation procedure is the same as with the training set with the exception of when the bad data is removed. The 1st test case represents the scenario where the physics-based filter is applied to the raw sensor data before any signal processing is done. The following is a summary of the procedure used to prepare the testing set.

- Import raw data into Matlab
- Incorporate desired degree of bad data
- Apply SMA Filter to the data (if necessary)
- Feature Mean Normalization (if necessary)
- Evaluate the TTF
• Health classification based on TTF and desired multiclass or binary classification
• Apply physics-based filter to remove bad data (if necessary)
• Make predictions on the test set with the desired trained model

The goal of this testing is to determine the optimal combination of SMA filter, feature mean normalization, multiclass or binary classification, and physics-based filtering that facilitates the best performance generalization of the trained models to the test set. Ultimately, with 48 models being tested on 9 variations of the test set, there are 432 sets of results to analyze.

2.7 Evaluating Algorithm Performance
As discussed in the previous section, the testing set contains skewed data. This means that it is not best practice to evaluate a trained algorithms performance on both the training set and testing set using prediction accuracy alone. Prediction accuracy is simply defined as the number of correct predictions divided by the number of predictions made overall, which is mathematically expressed as follows:

\[
\text{Accuracy} = \frac{\# \text{ correct predictions}}{\# \text{ predictions}}
\]

(61)

Consider the scenario in which there are 100 examples of data that are to be classified into two classes: the negative class \( (y = 0) \) and the positive class \( (y = 1) \). The negative class, of which there are 98 examples in the dataset, represents a transaction being classified as non-fraudulent. The positive class, of which there are 2 examples in the dataset, represents a transaction being classified as fraudulent. If this dataset is passed through an algorithm that is built to only classify transactions as non-fraudulent, it would result in that algorithm having 98% accuracy since it made correct predictions on the 98 negative examples and only made 2 incorrect predictions on the positive examples. 98% accuracy sounds excellent, but the reality is that the algorithm did
not correctly classify the examples that were most important, rendering the algorithm useless. In this thesis, the most important examples make up only 6.77% of the testing set. To mirror the fraudulent transaction, if a trained model made binary classifications on the C-MAPSS test set and classified each example as healthy, its accuracy would be 93.23%. Again, this number sounds desirable, but the algorithm is not actually doing its intended job. To that end, there are better parameters than accuracy to describe the performance of the trained machine learning models.

The precision associated with making predictions on a given class is defined as the ratio of true positives of that class to the sum of true positives and false positives of that class. In this thesis, precision asks the question: of all engines predicted unhealthy, what percentage are actually unhealthy? Precision, $Pr$, is mathematically expressed as follows:

$$
Pr = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false positives}} = \frac{\# \text{ true positives}}{\# \text{ predicted positive}}
$$

(62)

Precision is always a value between 0 and 1. While a high precision is always desirable, in the field of engine health management, engineers can stomach a precision less than 1. If some healthy engines are classified as unhealthy, this could mean the healthy engines condition is deteriorating and that engineers should keep an eye on it should further unhealthy predictions be made on new data. Precision, however, does not tell the whole story because it does not account for unhealthy engines that are mistakenly classified as healthy.

The recall associated with making predictions on a given class is defined as the ratio of true positives of that class to the sum of true positives and false negatives of that class. In this thesis, recall asks the question: of all engines that are actually unhealthy, what fraction did we correctly detect as being unhealthy? Recall, $Re$, is mathematically expressed as follows:
Like precision, recall is always a value between 0 and 1. Machine learning models are said to be doing well if they have both high precision and high recall [21]. This thesis proposes, though, that recall is more important in the field of engine health management, as it is important that unhealthy engines be classified as such. Mistakenly classifying an unhealthy engine as healthy can have disastrous consequences. Precision is still important, though, as a model that simply classifies all new examples as unhealthy in order to maintain perfect recall is useless. As such, for gas turbine health management, it is imperative to build machine learning models that find a balance between precision and recall.

In order to find a balance between precision and recall, it is desirable to find a way to mathematically express a model’s precision and recall with one value. The first measure that comes to mind is simply taking the average between the precision and recall for a given training or testing set. This is not a good measure, though. For example, consider a trained model that has a precision of 1 and a recall of 0 when asked to make predictions on a testing set. Further, consider a trained model that has a precision of 0.4 and a recall of 0.6 when asked to make predictions on a testing set. These models will have the same average of their precision and recall when the performance of the two models is actually vastly different. The most common way to amalgamate precision and recall values into one number is the F1 score [21]. The $F1$ score is mathematically expressed as follows:

$$Re = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false negatives}} = \frac{\# \text{ true positives}}{\# \text{ actual positives}}$$
\[ F1 = 2 \frac{PrRe}{Pr + Re} \]  \hspace{1cm} (64)

In the previous example, the model who’s precision is 1 and recall is 0 will have an F1 score of 0. The model whose precision is 0.4 and recall is 0.6 will have an F1 score of 0.48, which is more representative of the models’ performance using one value on a scale from 0 to 1. A 3D plot of precision, recall, and F1 score is shown below in Figure 2-15

![3D Plot and 2D slice of F1 Score Vs. Precision and Recall](image)

**Figure 2-15: 3D Plot and 2D slice of F1 Score Vs. Precision and Recall [29]**

From the figure, it is clear that the F1 score is a useful metric that provides a balance between precision and recall that makes it a desirable performance benchmarking value. When both precision and recall are high, F1 score is high. When one of precision or recall lags behind the other, that is reflected in the metric. If one of precision or recall is very small, the F1 score will be very small. In order to evaluate the F1 score for my multiclass models, all examples that aren’t classified as urgent are simply considered as one class, regardless of whether they are
classified as long, short, or medium. This allows comparison of precision, recall, and $F_1$ scores for predicting examples as urgent/unhealthy between the multiclass and binary models.

In the following Results and Discussion chapter, the performance of the trained models from Table 2-11 is evaluated on both the training and testing set. To evaluate training set performance, the overall accuracy as well as the precision and recall of each model on the urgent/unhealthy class of the training set is examined. In addition, scatter plots showing correct and incorrect predictions, provided as output from Matlab’s Classification Learner, are used to study the strengths and weaknesses of each model when it comes to making predictions on the training set. These same tools are used to evaluate the models’ performance on the testing sets. Through this analysis, the efficacy of the novel physics-based filter is displayed and the best combination of processing and classification steps (SMA filter, feature normalization, physics-based filter, binary/multiclass classification) is presented.
3 Results and Discussion

3.1 Training Set Results

This section focuses strictly on sensor data pertaining to the HPC and LPT outlet temperature since those values are manipulated in the testing set. Further, there are 14 sensors in the dataset, so it is not feasible to visualize all of them. The first component to analyze is how the combinations of SMA filter, feature mean normalization and multiclass/binary health classification affect the data in the training set. This is best visualized by a scatter plot of HPC outlet temperature vs. LPT outlet temperature where each data point is colored based on its classification. Figures 3-1 and 3-2 display these scatter plots for the all engine training sets. The scatter plots for the 10 engine sample training sets are omitted as they follow similar trends to their all engine counterparts.

![Figure 3-1: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered, Unnormalized, Multiclass Training Set](image-url)
From Figure 3-1 it is clear that there is some overlap between the classes. However, it is important to remember that this plot contains only two of the fourteen sensors in the training set, so there are other differentiating factors within the remaining data. Since this is unfiltered data, the hope is that the SMA filter will serve to create more separation between the data points. Still, though, there is a clear trend in temperature values from the long class to the urgent class.

![Figure 3-2: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered, Unnormalized, Binary Training Set](image)

When switching from multiclass classification to binary classification, a clear separation between healthy and unhealthy training examples is revealed. There is certainly still some overlap as engines transition from healthy to unhealthy, though. Table 3-1 below compares the accuracy, precision, recall, and $F_1$ scores of all the models trained with this training set, as well as the models trained with data from 10 engines from this training set.
Table 3-1: Summary of Training Set Results For Models Trained on Unfiltered, Unnormalized Data

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>Classification</th>
<th>Method</th>
<th>EngineList</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>62.8%</td>
<td>0.870</td>
<td>0.850</td>
<td>0.860</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>No</td>
<td>No</td>
<td>58.8%</td>
<td>0.710</td>
<td>0.850</td>
<td>0.774</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>62.5%</td>
<td>0.840</td>
<td>0.810</td>
<td>0.825</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>No</td>
<td>No</td>
<td>63.5%</td>
<td>0.820</td>
<td>0.730</td>
<td>0.772</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>92.7%</td>
<td>0.780</td>
<td>0.910</td>
<td>0.840</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>No</td>
<td>No</td>
<td>81.8%</td>
<td>0.300</td>
<td>0.890</td>
<td>0.449</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>64.0%</td>
<td>0.820</td>
<td>0.890</td>
<td>0.854</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>No</td>
<td>No</td>
<td>63.0%</td>
<td>0.820</td>
<td>0.860</td>
<td>0.840</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>56.4%</td>
<td>0.930</td>
<td>0.730</td>
<td>0.818</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>No</td>
<td>No</td>
<td>56.1%</td>
<td>0.930</td>
<td>0.720</td>
<td>0.812</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>93.1%</td>
<td>0.820</td>
<td>0.890</td>
<td>0.854</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>No</td>
<td>No</td>
<td>92.1%</td>
<td>0.810</td>
<td>0.860</td>
<td>0.834</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first detail that comes to mind when looking at Table 3-1 is that the binary classification models have a much higher overall prediction accuracy than the multiclass models. As previously discussed, this is highly misleading. Referring to Table 2-7, the multiclass training set contains data distributed somewhat evenly amongst the classes (with the exception of the long class). On the other hand, referring to Table 2-8, the binary training set is just over 75% healthy examples. The accuracy results indicate that the multiclass models without the SMA filter and feature mean normalization struggle to consistently classify all four classes. However, looking at the precision, recall, and F1 scores in Table 3-1, it is clear that the multiclass models perform similarly to the binary models when it comes to predicting the urgent/unhealthy class.
Ideally, a desirable predictive model for gas turbine health management has excellent recall with reasonable precision. The model with the highest recall is the Fine Gaussian SVM trained on a binary training set, although this model has a less-than-ideal precision score. The model with the highest F1 score is the Fine Gaussian SVM trained on a multiclass training set with the 1v1 method. This model provides a great balance between precision and recall, but tipping the scales in favor of recall without sacrificing the F1 score is slightly more desirable. As such, the Linear SVM trained on the multiclass training set using the 1v1 method and the Linear SVM trained on the binary training set provide the best balance between the scores while maintaining a high recall score. It is also worth pointing out that the Fine Gaussian SVM trained on the binary training set made using only examples from 10 engines has by far the worst performance on its training set in terms of precision, recall and F1 score. This is likely because the Gaussian SVM aims to fit a complex non-linear decision boundary around the data and there are simply not enough examples in the 10 engine training set to define a suitable binary decision boundary [21]. While this model has a high recall, its low precision implies that it classifies the majority of examples as unhealthy, which makes it useless in practice as there would be too many engines falsely classified as unhealthy.
Figure 3-3: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Unfiltered, Normalized, Multiclass Training Set

Normalizing the unfiltered multiclass training data set does not do much to change the distribution of points in the HPC vs. LPT outlet temperature distribution plot. There is still overlap between the classes although the short and urgent classes seem to be more separated from the long and medium classes. This implies that these temperatures start to deviate more and more from their average as degradation sets in.
Similarly, normalizing the unfiltered binary training data set does not do much to change the distribution of points in the HPC vs. LPT outlet temperature distribution plot. In fact, there are more examples classified as healthy that extend into the unhealthy cluster of data points. This implies that these temperatures start to deviate from the average before the 50 cycles before failure cutoff that defines the difference between these classifications. Table 3-2 below compares the accuracy, precision, recall, and $F1$ scores of all the models trained with this training set, as well as the models trained with data from 10 engines from this training set.
Table 3-2: Summary of Training Set Results For Models Trained on Unfiltered, Normalized Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification</th>
<th>Method</th>
<th>Engine/Jst</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>66.5%</td>
<td>0.920</td>
<td>0.840</td>
<td>0.878</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>53.1%</td>
<td>0.540</td>
<td>0.870</td>
<td>0.666</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>66.9%</td>
<td>0.890</td>
<td>0.750</td>
<td>0.814</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>63.4%</td>
<td>0.760</td>
<td>0.910</td>
<td>0.828</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>89.6%</td>
<td>0.640</td>
<td>0.910</td>
<td>0.751</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>Yes</td>
<td>78.6%</td>
<td>0.140</td>
<td>0.930</td>
<td>0.243</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>70.9%</td>
<td>0.870</td>
<td>0.900</td>
<td>0.885</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>69.3%</td>
<td>0.850</td>
<td>0.900</td>
<td>0.874</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>61.2%</td>
<td>0.960</td>
<td>0.790</td>
<td>0.867</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>64.5%</td>
<td>0.950</td>
<td>0.790</td>
<td>0.863</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.870</td>
<td>0.900</td>
<td>0.885</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.860</td>
<td>0.910</td>
<td>0.884</td>
<td></td>
</tr>
</tbody>
</table>

Similarly with the unfiltered, unnormalized training set, the binary classification models have a much higher overall prediction accuracy than the multiclass models. Again, the accuracy results indicate that the multiclass models without the SMA filter with feature mean normalization still struggle to consistently classify all four classes. Looking at the precision, recall, and F1 scores in Table 3-2, The Linear SVM models consistently outperform their Gaussian SVM counterparts when it comes to predicting the urgent/unhealthy class. This implies that a simple, linear decision boundary is most suitable for these training sets. The model with the highest recall is the Fine Gaussian SVM trained on a binary training set. However, this model is actually the worst performing because it classifies nearly all the
examples as healthy, as indicated by the low precision and low F1 scores. This lends credence to the idea that these models cannot be evaluated using only one performance indicator alone. The models with the next highest recall scores are the Linear SVM trained on a multiclass training set with the 1v1 method and the Linear SVM trained on a binary training set. These models provide an exceptional recall while also having a precision that is suitably high. Further, these models also have the highest F1 score of the models trained on these training sets. Comparing the results in Table 3-2 to the results in Table 3-1, feature mean normalization increases the F1 scores of all of the Linear SVM models. Overall, the Linear SVM model trained on the multiclass training set with 1v1 method and the Linear SVM model trained on the binary training set are the best performing models trained on unfiltered data.
Applying the SMA filter to the training dataset makes the data points fall within a much tighter range than the data points in the unfiltered training set. This is expected, as the mixed distribution noise added to the sensed outputs of the C-MAPSS simulation serves to create more outliers within the dataset. The SMA filter creates more vertical distance between the classifications, which should help the models performance in classifying the training examples.
When switching from multiclass classification to binary classification, there is again a more clear separation between healthy and unhealthy training examples. While there are still some healthy examples that appear to bleed into the unhealthy territory, the SMA filter creates some vertical distance earlier on in the transition from healthy to unhealthy. Again, it is important to remember that the dataset contains data from twelve other sensors whose data is being affected by the SMA filter. Table 3-3 below compares the accuracy, precision, recall, and F1 scores of all the models trained with this training set, as well as the models trained with data from 10 engines from this training set.

Figure 3-6: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Filtered, Unnormalized, Binary Training Set
**Table 3-3: Summary of Training Set Results For Models Trained on Filtered, Unnormalized Data**

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>EngineList</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>84.2%</td>
<td>0.890</td>
<td>0.940</td>
<td>0.914</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
<td>94.4%</td>
<td>0.960</td>
<td>0.950</td>
<td>0.955</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>78.0%</td>
<td>0.880</td>
<td>0.940</td>
<td>0.909</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
<td>92.5%</td>
<td>0.960</td>
<td>0.950</td>
<td>0.955</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>95.9%</td>
<td>0.880</td>
<td>0.950</td>
<td>0.914</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>Yes</td>
<td>no</td>
<td>98.2%</td>
<td>0.940</td>
<td>0.980</td>
<td>0.960</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>64.8%</td>
<td>0.830</td>
<td>0.890</td>
<td>0.859</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
<td>65.2%</td>
<td>0.830</td>
<td>0.870</td>
<td>0.850</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>56.1%</td>
<td>0.930</td>
<td>0.740</td>
<td>0.824</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>10sample</td>
<td>Yes</td>
<td>no</td>
<td>60.7%</td>
<td>0.920</td>
<td>0.730</td>
<td>0.814</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>93.3%</td>
<td>0.820</td>
<td>0.900</td>
<td>0.858</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>Yes</td>
<td>no</td>
<td>92.3%</td>
<td>0.810</td>
<td>0.870</td>
<td>0.839</td>
</tr>
</tbody>
</table>

The results of the models trained on filtered data are strikingly different than those of the models trained on unfiltered data. In terms of F1 score, The Fine Gaussian SVM models actually
perform better than the Linear SVM models across the board. The three models with the highest F1 scores are all three of the Fine Gaussian SVM models trained with data from 10 engines, while the F1 scores from the Fine Gaussian SVM models trained with data from all engines lags slightly behind. This implies that the data from those 10 engines, after being ran through the SMA filter, is polished enough to build nearly perfect decision boundaries. This instills caution because, as discussed in the Machine Learning section, this may imply that the model has overfit the training set; it is trained to make predictions near perfect predictions on the training set but may not generalize well to new examples [21]. When a model overfits the training set it is referred to as having high variance. As previously discussed, one way to improve the performance on new examples of a trained model with high variance is to retrain the model with more training examples. This exact case is represented by the Fine Gaussian SVM trained on the binary training set with data from all engines. In Tables 3-1 and 3-2, there is always one model that lags behind the rest and struggles to make predictions on the training set. In Table 3-3 there are no such models. This implies that the SMA filter is effective in creating separation between the data points in each class, which makes it easier to train a well-performing machine learning model. Ultimately, the most desirable models from Table 3-3 are the Fine Gaussian SVMs trained on the multiclass training set using the 1v1 method and the binary training set with data from all engines.
Figure 3-7: Scatter Plot of Total HPC and LPT Outlet Temperatures For the Filtered, Normalized, Multiclass Training Set

Comparing Figure 3-7 to Figure 3-5, feature mean normalizing the filtered training set data makes the HPC and LPT total outlet temperatures lie within a tighter range. Further, comparing Figure 3-7 to Figure 3-3, applying the SMA filter before feature mean normalizing the data creates much more separation between the classes both horizontally and vertically. This increased separation should improve the performance of these trained models compared to their unfiltered counterparts.
Switching from multiclass to binary classification for the filtered and normalized training set has the same effect as it did for the filtered and unnormalized training set. Since there are only two classes, there is a more clear separation between healthy and unhealthy training examples, so the accuracy of the trained models should be higher compared to the multiclass classification models. Comparing Figure 3-8 to Figure 3-4 shows the importance of the SMA filter. There is more vertical and horizontal distance between the feature mean normalized data when the SMA filter is applied beforehand. Table 3-4 below compares the accuracy, precision, recall, and F1 scores of all the models trained with this training set, as well as the models trained with data from 10 engines from this training set.
Table 3-4: Summary of Training Set Results for Models Trained on Filtered, Normalized Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>81.3%</td>
<td>0.960</td>
<td>0.920</td>
<td>0.940</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1 10sample</td>
<td>Yes</td>
<td>Yes</td>
<td>85.5%</td>
<td>0.990</td>
<td>0.950</td>
<td>0.970</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>82.2%</td>
<td>0.940</td>
<td>0.930</td>
<td>0.935</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1vAll 10sample</td>
<td>Yes</td>
<td>Yes</td>
<td>89.9%</td>
<td>0.960</td>
<td>0.960</td>
<td>0.960</td>
<td></td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>97.1%</td>
<td>0.930</td>
<td>0.950</td>
<td>0.940</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary 10 sample</td>
<td>Yes</td>
<td>Yes</td>
<td>98.4%</td>
<td>0.950</td>
<td>0.980</td>
<td>0.965</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>74.9%</td>
<td>0.900</td>
<td>0.910</td>
<td>0.905</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1 10sample</td>
<td>Yes</td>
<td>Yes</td>
<td>77.8%</td>
<td>0.900</td>
<td>0.940</td>
<td>0.920</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>64.7%</td>
<td>0.970</td>
<td>0.820</td>
<td>0.889</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1vAll 10sample</td>
<td>Yes</td>
<td>Yes</td>
<td>70.3%</td>
<td>0.960</td>
<td>0.840</td>
<td>0.896</td>
<td></td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>95.4%</td>
<td>0.870</td>
<td>0.900</td>
<td>0.885</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary 10 sample</td>
<td>Yes</td>
<td>Yes</td>
<td>96.3%</td>
<td>0.860</td>
<td>0.910</td>
<td>0.884</td>
<td></td>
</tr>
</tbody>
</table>

Similarly to the results in Table 3-3, in terms of F1 score, the Fine Gaussian SVM models perform better than the Linear SVM models across the board. This further supports that the SMA filtered data facilitates the creation of suitable complex non-linear decision boundaries for this data. This is another notion that should instill caution, though, because a complex non-linear decision boundary may fit the training data perfectly but may also struggle to generalize to new examples [21]. This observation is explored and discussed in the Testing Set section of this chapter. By comparing Tables 3-1, 3-2, 3-3, and 3-4, it is clear that the Fine Gaussian SVM models perform better on the filtered training data, and the Linear SVM models perform better on the unfiltered training data. By comparing Tables 3-3 and 3-4, it is evident that the F1 scores
in Table 3-4 are higher than those in Table 3-3 across the board. This implies that the feature mean normalization of the filtered data facilitates better performance on the training set. Out of all the models trained on filtered, normalized training data, the most desirable models are the Fine Gaussian SVMs trained on the multiclass training set using the 1v1 method and the binary training set with data from all engines.

Table 3-5 below shows the best trained models whose performance on the testing sets is evaluated in the next section.

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Norm</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>64.0%</td>
<td>0.820</td>
<td>0.890</td>
<td>0.854</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
<td>63.0%</td>
<td>0.820</td>
<td>0.860</td>
<td>0.840</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>93.1%</td>
<td>0.820</td>
<td>0.890</td>
<td>0.854</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>no</td>
<td>92.1%</td>
<td>0.810</td>
<td>0.860</td>
<td>0.834</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>62.8%</td>
<td>0.870</td>
<td>0.850</td>
<td>0.860</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
<td>58.8%</td>
<td>0.710</td>
<td>0.850</td>
<td>0.774</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>70.9%</td>
<td>0.870</td>
<td>0.900</td>
<td>0.885</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>69.3%</td>
<td>0.850</td>
<td>0.900</td>
<td>0.874</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.870</td>
<td>0.900</td>
<td>0.885</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.860</td>
<td>0.910</td>
<td>0.884</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>66.5%</td>
<td>0.920</td>
<td>0.840</td>
<td>0.878</td>
</tr>
</tbody>
</table>
The next section of the Results and Discussion chapter explores how the best performing trained models perform when asked to make predictions on the previously-unseen testing sets.

To display the importance of the number of training examples, the performance of the all engine and 10 sample versions of the best trained models is analyzed. Since the Linear SVM models performed better on the unfiltered training data, the best performing Fine Gaussian SVM model
trained on unfiltered data is also used in the next section to show how both SVM schemes generalize to new examples. Since the Fine Gaussian SVM models performed better on the filtered training data, the best performing Linear SVM model trained on filtered data is also used in the next section for the same purpose. In both cases, the multiclass models trained using the 1v1 method are the best. For all multiclass training cases, the models trained using the 1v1 method perform better than those trained using the 1vAll method. This is the expected result because, as previously discussed in the machine learning section, the 1v1 method builds separate classifiers to classify each class against one of the others, whereas the 1vAll method only builds separate classifiers to classify one class against all of the classes [21].

3.2 Testing Set Results
This section studies how the models in Table 3-5 performs on the 9 test cases discussed in the Methodology section. For convenience, the 9 test cases are laid out below.

1. Test set only contains good data
2. Test set contains 1000 random points of harshly faulty data
3. Physics-based filter applied to #2 to remove bad data points
4. Test set contains 1000 random points of mildly faulty data
5. Physics-based filter applied to #4 to remove bad data points
6. Test set contains harshly faulty data on the even units
7. Physics-based filter applied to #6 to remove bad data points
8. Test set contains mildly faulty data on the even units
9. Physics-based filter applied to #8 to remove bad data points
3.2.1 Test #1: Test On Good Data

The well-performing trained models are tested using the testing set from Test #1. This establishes a baseline for each model’s performance when asked to make predictions on new data. Further, as previously mentioned, this test case represents the scenario where the physics-based filter is applied directly to the raw sensor measurements and has either found no bad data or removed all the bad data before any signal processing has occurred. Figures 3-9 through 3-16 show the scatter plots of HPC and LPC total outlet temperature testing set data for each combination of SMA filter, feature mean normalization, and multiclass/binary classification.

![Figure 3-9: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set](image)
Figure 3-10: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set

Figure 3-11: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Unfiltered, Normalized, Multiclass Testing Set
Figure 3-12: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Unfiltered, Normalized, Binary Testing Set

Figure 3-13: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set
Figure 3-14: Scatter Plot of Total HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set

Figure 3-15: Scatter Plot of Total HPC and LPT Total Outlet Temperatures for the Filtered, Normalized, Multiclass Testing Set
Comparing Figures 3-9 through 3-16 to their training set counterparts in Figures 3-1 through 3-8, it is clear that there are some differences between the training set and the testing set. First, as previously mentioned, the testing set does not contain complete run-to-failure data. Rather, it contains random samples of consecutive flights for the engines. These flights could all be when a given engine is healthy, or they could be on the transition from healthy to unhealthy. This results in the skewed nature of the testing set, with only 6.77% of examples being classified as urgent/unhealthy. Due to this, there is a much less clear boundary between the healthy examples and the unhealthy/urgent examples in Figures 3-9 through 3-16. Applying the SMA filter to the testing data creates a much clearer boundary between classifications, just as it did with the training set. Feature mean normalizing the testing set, though, makes the boundary between healthy and unhealthy examples less clear. Many of the testing examples end up falling
within one large cluster after normalization, which highlights the noise in the testing set and also that many of the unhealthy/urgent examples are closer to the high end of the classification threshold (<50 cycles before failure). For a trained machine learning algorithm, there tends to be higher error when making predictions on new examples compared to making predictions on the training set [21]. Given the nature of the scatter plots in Figures 3-9 through 3-16, there is no expectation that the trained models perform as well on the testing set as they did on the training set. As such, a performance threshold is implemented such that any of the well-performing trained models that fail to generalize well to new examples are not tested any further. This threshold aims to emulate a real-life model selection setting. Since it is not feasible to generate more training data, only models that show adequate ability to make predictions on new, good data move on to the next test cases. This threshold is arbitrarily determined based on the performance of the trained models on the testing set from Test #1. Table 3-6 on the next page shows the results of Test #1.
Table 3-6: Summary of Results for All Well-Performing Trained Models on the Testing Set From Test #1

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Training Set</th>
<th>Good Data Testing Set Test #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>64.0%</td>
<td>0.820</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
<td>63.0%</td>
<td>0.820</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>63.1%</td>
<td>0.820</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>no</td>
<td>79.1%</td>
<td>0.810</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>94.5%</td>
<td>0.870</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
<td>94.5%</td>
<td>0.870</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>70.9%</td>
<td>0.870</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>69.3%</td>
<td>0.850</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.870</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>no</td>
<td>Yes</td>
<td>94.5%</td>
<td>0.860</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>Yes</td>
<td>66.5%</td>
<td>0.920</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>Yes</td>
<td>53.1%</td>
<td>0.540</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>yes</td>
<td>no</td>
<td>84.2%</td>
<td>0.890</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>yes</td>
<td>no</td>
<td>94.4%</td>
<td>0.960</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>yes</td>
<td>no</td>
<td>95.9%</td>
<td>0.880</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>yes</td>
<td>no</td>
<td>98.2%</td>
<td>0.940</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>64.8%</td>
<td>0.830</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>no</td>
<td>no</td>
<td>65.2%</td>
<td>0.830</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>yes</td>
<td>yes</td>
<td>81.3%</td>
<td>0.960</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>yes</td>
<td>yes</td>
<td>85.5%</td>
<td>0.990</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>yes</td>
<td>yes</td>
<td>97.1%</td>
<td>0.930</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>10 sample</td>
<td>yes</td>
<td>yes</td>
<td>98.4%</td>
<td>0.950</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>yes</td>
<td>yes</td>
<td>74.9%</td>
<td>0.900</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>10sample</td>
<td>yes</td>
<td>yes</td>
<td>77.8%</td>
<td>0.900</td>
</tr>
</tbody>
</table>

Accuracy | Recall | Precision | F1 Score | Accuracy | Recall | Precision | F1 Score |
---|---|---|---|---|---|---|---|
64.0% | 0.820 | 0.890 | 0.854 | 59.1% | 0.611 | 0.807 | 0.695 |
63.0% | 0.820 | 0.860 | 0.840 | 57.1% | 0.517 | 0.834 | 0.638 |
93.1% | 0.820 | 0.890 | 0.854 | 96.4% | 0.602 | 0.817 | 0.693 |
92.1% | 0.810 | 0.860 | 0.834 | 96.1% | 0.507 | 0.867 | 0.640 |
62.8% | 0.870 | 0.850 | 0.860 | 52.5% | 0.696 | 0.736 | 0.715 |
58.8% | 0.710 | 0.850 | 0.774 | 43.1% | 0.447 | 0.773 | 0.566 |
70.9% | 0.870 | 0.900 | 0.885 | 41.1% | 0.956 | 0.337 | 0.498 |
69.3% | 0.850 | 0.900 | 0.874 | 41.6% | 0.949 | 0.332 | 0.492 |
94.5% | 0.870 | 0.900 | 0.885 | 87.0% | 0.956 | 0.338 | 0.499 |
94.5% | 0.860 | 0.910 | 0.884 | 86.8% | 0.949 | 0.333 | 0.493 |
66.5% | 0.920 | 0.840 | 0.878 | 32.4% | 0.921 | 0.307 | 0.461 |
53.1% | 0.540 | 0.870 | 0.666 | 34.2% | 0.345 | 0.541 | 0.421 |
84.2% | 0.890 | 0.940 | 0.914 | 51.9% | 0.635 | 0.763 | 0.693 |
94.4% | 0.960 | 0.950 | 0.955 | 44.8% | 0.676 | 0.582 | 0.625 |
95.9% | 0.880 | 0.950 | 0.914 | 96.5% | 0.623 | 0.823 | 0.709 |
98.2% | 0.940 | 0.980 | 0.960 | 95.5% | 0.492 | 0.757 | 0.596 |
64.8% | 0.830 | 0.890 | 0.859 | 60.3% | 0.626 | 0.825 | 0.712 |
65.2% | 0.830 | 0.870 | 0.850 | 56.1% | 0.525 | 0.795 | 0.632 |
81.3% | 0.960 | 0.920 | 0.940 | 34.6% | 0.989 | 0.298 | 0.458 |
85.5% | 0.990 | 0.950 | 0.970 | 24.4% | 0.989 | 0.139 | 0.244 |
97.1% | 0.930 | 0.950 | 0.940 | 93.4% | 0.885 | 0.509 | 0.646 |
98.4% | 0.950 | 0.980 | 0.965 | 94.4% | 0.571 | 0.593 | 0.582 |
74.9% | 0.900 | 0.910 | 0.905 | 41.8% | 0.968 | 0.283 | 0.438 |
77.8% | 0.900 | 0.940 | 0.920 | 40.6% | 0.971 | 0.272 | 0.425 |
The most noteworthy result in Table 3-6 is that, for each model, the F1 scores are lower on the testing set than they are on the training set. This performance drop is to be expected, although not to the extent experienced by some of the models [21]. The lack of clear boundaries between classifications in the testing set scatter plots in Figures 3-9 through 3-16 explains the startlingly diminished performance from some of the algorithms and, in particular, those trained and tested using feature mean normalized data. Regardless, all the trained models suffer from high variance; they overfit the training set. One way to fix high variance is to find more training examples with which to retrain the models on. However, given the limitation of using only the C-MAPSS dataset and embracing the reality of the scarcity of run-to-failure datasets, there are no remaining training examples with which to train the models. The effect of the lack of training examples is best evidenced by examining the results of all of the models trained on training sets with examples from only 10 engines. The F1 scores of those models are less than their all engine counterparts across the board. High variance notwithstanding, there are still some trained models that exhibit adequate generalization performance and advance to the next test cases. All trained models in Table 3-6 must meet the following performance thresholds on Test #1:

- F1 score greater than 0.65 OR
- Recall greater than 0.8 AND Precision greater than 0.5

Table 3-7 on the next page shows the remaining models that meet this performance threshold.
Table 3-7: Summary of Trained Models That Demonstrate Adequate Performance on the Good Data Testing Set from Test #1

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Training Set</th>
<th>Good Data Test #1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>64.0%</td>
<td>0.820</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>93.1%</td>
<td>0.820</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>62.8%</td>
<td>0.870</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>84.2%</td>
<td>0.890</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>95.9%</td>
<td>0.880</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>64.8%</td>
<td>0.830</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>97.1%</td>
<td>0.930</td>
</tr>
</tbody>
</table>
It is interesting to note that only one of the models trained on feature mean normalized data performs adequately on the testing set from Test #1. Looking at Table 3-6, all feature mean normalized models tend to categorize most of the testing examples as urgent/unhealthy. As previously mentioned, this can be stomached to an extent, but only if the precision of those models is reasonable. There was only one model, the Fine Gaussian SVM trained on SMA filtered and feature mean normalized data, that had a reasonable enough precision value to push the F1 score near 0.65. From here on, all the remaining models in Table 3-7 are tested on the remaining test cases. It also important to note that none of the unfiltered, feature mean normalized models performed adequately on the testing set. As such, the remaining test matrix only contains 5 combinations of SMA filter, feature mean normalization, and multiclass/binary classification: no/no/multiclass, no/no/binary, yes/no/multiclass, yes/no/binary, yes/yes/binary.

3.2.2 Test #2: Test With 1000 Harshly Faulty Data Points in Testing Set

The testing set in Test #2 contains 1000 random data points whose HPC and LPT total outlet temperatures are harshly faulty. This test case represents the scenario where there is a faulty component along the data transmission path from the sensor to the DAQ computer. In such a scenario, the faulty component may intermittently disrupt the signal it is sending, thus creating random bad data points. Half of the 1000 testing examples have their HPC total outlet temperature multiplied by 0.8 and the other half have their LPT total outlet temperature multiplied by 0.5. These constant factors increase the adiabatic efficiency of the compressor and turbine sections of the engines to about 1.25. Figures 3-17 through 3-21 below show the scatter plots of the HPC vs. LPT total outlet temperatures for each of the remaining combinations of SMA filter, feature mean normalization, and multiclass/binary classification.
Figure 3-17: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Harshly Faulty Examples

Figure 3-18: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Harshly Faulty Examples
Figure 3-19: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Harshly Faulty Examples

Figure 3-20: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Harshly Faulty Examples
Figure 3-21: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with 1000 Harshly Faulty Examples

Comparing Figures 3-17 through 3-21 to each other, it is clear that there are some differences in how the bad data is treated by the filtering and normalizing signal processing methods. Figures 3-17 and 3-18 are unfiltered, which means that the value of any given training example does not affect the value of any other training example. This facilitates the creation of three groups of points: one group whose HPC and LPT total outlet temperature data is untouched, one group whose HPC total outlet temperature is harshly faulty and LPT total outlet temperature is untouched, and one group whose LPT total outlet temperature is harshly faulty and HPC total outlet temperature is left untouched. The data in Figures 3-19 and 3-20 are run through the SMA filter, so the bad data points effected the good points. This causes the sprinkling effect seen in those figures, where the data points closer to the directly faulty data are pulled closer and closer to the extremities of the three groupings. The data in Figure 3-21 is both
ran through the SMA filter and feature mean normalized, which creates a more continuous
distribution of faulty data. Further, there is no clear separation between the healthy and
unhealthy examples in Figure 3-21. All testing sets in Test #2 ask the trained models to make
predictions on harshly faulty data, so the models’ performance is expected to be greatly
diminished compared to the Test #1 results. Table 3-8 on the next page shows the results of Test
#2 compared to Test #1.
Table 3-8: Summary of Model Performance on Testing Set with 1000 Harshly Faulty Examples Compared To Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test#1</th>
<th>1000 Harshly Faulty Test #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
</tr>
</tbody>
</table>
The F1 scores for each model decrease when asked to make predictions on the new testing set with 1000 harshly faulty data points. The F1 scores of the Linear SVM models do not decrease as much as those of the Fine Gaussian SVM models. This is because the Linear SVMs create a linear decision boundary whose performance when asked to make predictions on data faulty by a constant factor will not be as impeded as the complex non-linear decision boundary of the Fine Gaussian SVM. With that in mind, it is interesting to point out that the Fine Gaussian SVM trained on the SMA filtered, unnormalized, binary training set does not see the same drop in performance as the other Fine Gaussian SVM models. This implies that the combination of SMA filter, no feature mean normalization, and binary training data yields the best performing model. Still, though, all models in Table 3-8 clearly perform worse when asked to make predictions on faulty data, as expected.

3.2.3 Test #3: Test With 1000 Harshly Faulty Data Points Using Physics-Based Filter

The testing set in Test #3 is ran through the physics-based filter after all the data is made faulty and signal processing is done. Any remaining data points that violate the laws of thermodynamics are removed from the dataset. For the unfiltered testing sets, the physics filter removes all 1000 points that are faulty. For the filtered testing sets, the physics filter does not catch all the faulty data points because most have been smoothed to below the removal threshold due to the good data points surrounding them. Tables 3-9 through 3-12 below show the classification distributions of the remaining examples in the unfiltered and filtered testing sets for both multiclass and binary classification.
Table 3-9: Summary of Classification Distribution of Unfiltered Multiclass Testing Set with 1000 Harshly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1812</td>
<td>14.98%</td>
</tr>
<tr>
<td>Medium</td>
<td>5527</td>
<td>45.69%</td>
</tr>
<tr>
<td>Short</td>
<td>3933</td>
<td>32.51%</td>
</tr>
<tr>
<td>Urgent</td>
<td>824</td>
<td>6.81%</td>
</tr>
</tbody>
</table>

Table 3-10: Summary of Classification Distribution of Unfiltered Binary Testing Set with 1000 Harshly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>11272</td>
<td>93.19%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>824</td>
<td>6.81%</td>
</tr>
</tbody>
</table>

Table 3-11: Summary of Classification Distribution of Filtered Multiclass Testing Set with 1000 Harshly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1907</td>
<td>14.96%</td>
</tr>
<tr>
<td>Medium</td>
<td>5789</td>
<td>45.40%</td>
</tr>
<tr>
<td>Short</td>
<td>4173</td>
<td>32.73%</td>
</tr>
<tr>
<td>Urgent</td>
<td>882</td>
<td>6.92%</td>
</tr>
</tbody>
</table>

Table 3-12: Summary of Classification Distribution of Filtered Binary Testing Set with 1000 Harshly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>11869</td>
<td>93.08%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>882</td>
<td>6.92%</td>
</tr>
</tbody>
</table>

There are 12096 testing examples in the unfiltered testing set and 12751 testing examples in the filtered testing set. This shows just how many data points that the SMA filter is able to smooth under the removal threshold. Only 345 of the 1000 faulty data points are removed in the filtered testing sets. Figures 3-22 through 3-26 show the scatter plots of the testing sets that each model is asked to make predictions on.
Figure 3-22: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter

Figure 3-23: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter
Figure 3-24: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter

Figure 3-25: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Harshly Faulty Examples After Being Ran Through the Physics Filter
Figures 3-22 through 3-26 show the effectiveness of the physics-based filter on datasets with various levels of signal processing. The physics filter returns both the multiclass and binary unfiltered testing sets back to their original state, albeit with 1000 less data points. Looking at Figures 3-22 and 3-23, there is no way to tell that the data was faulty in the first place. For datasets where the SMA filter is applied, though, the physics-based filter can only remove the most extreme outliers. Comparing Figure 3-24 and 3-25 to Figures 3-19 and 3-20, after the physics-based filter is applied, the data points lie within a much tighter range although they are still far from their original state. Comparing Figure 3-26 to Figure 3-21, after the physics-based filter is applied, the extreme outliers are removed. The minimums of the x and y axes in Figure 3-26 are -5 and -4, respectively, compared to both being at -8 in Figure 3-21.
Since all bad data points are removed in the unfiltered testing sets, the performance of the models trained on unfiltered training data is expected to approach the performance of those models on Test #1, the testing set with only good data. The models trained on filtered data, though, should not return to their Test #1 performance because faults still exist within the testing set for Test #3. Table 3-13 on the next page compares the results of Test #3 to those of both Test #2 and Test #1.
Table 3-13: Summary of Model Performance on Testing Set with 1000 Harshly Faulty Examples Ran Through the Physics-based Filter Compared to Performance on Testing Set with 1000 Harshly Faulty Examples and Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
<td>0.807</td>
<td>0.695</td>
<td>58.0%</td>
<td>0.568</td>
<td>0.810</td>
<td>0.668</td>
<td>58.9%</td>
<td>0.610</td>
<td>0.810</td>
<td>0.696</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.817</td>
<td>0.693</td>
<td>96.2%</td>
<td>0.560</td>
<td>0.818</td>
<td>0.665</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.818</td>
<td>0.694</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
<td>0.736</td>
<td>0.715</td>
<td>49.0%</td>
<td>0.710</td>
<td>0.355</td>
<td>0.473</td>
<td>52.5%</td>
<td>0.688</td>
<td>0.733</td>
<td>0.710</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
<td>0.763</td>
<td>0.693</td>
<td>36.7%</td>
<td>0.726</td>
<td>0.135</td>
<td>0.228</td>
<td>37.6%</td>
<td>0.724</td>
<td>0.145</td>
<td>0.242</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
<td>0.823</td>
<td>0.709</td>
<td>96.4%</td>
<td>0.555</td>
<td>0.859</td>
<td>0.674</td>
<td>96.3%</td>
<td>0.554</td>
<td>0.858</td>
<td>0.673</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
<td>0.825</td>
<td>0.712</td>
<td>57.2%</td>
<td>0.456</td>
<td>0.849</td>
<td>0.593</td>
<td>57.1%</td>
<td>0.458</td>
<td>0.849</td>
<td>0.595</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
<td>0.646</td>
<td>91.9%</td>
<td>0.413</td>
<td>0.405</td>
<td>0.409</td>
<td>91.7%</td>
<td>0.415</td>
<td>0.405</td>
<td>0.410</td>
</tr>
</tbody>
</table>
As expected, the precision, recall, and F1 scores on Test #3 of the models trained and tested on unfiltered data returned to those of Test #1. This makes sense because their testing sets for Test #3 are the same as Test #1, except that the 1000 harshly faulty data points are removed. The precision, recall, and F1 scores on Test #3 of the models trained and tested on filtered data, though, remained about the same as on Test #2 even after the physics-based filter is applied. Even after removing the data points that directly violate the laws of thermodynamics, the performance is essentially unchanged. Referring back to the class distributions in Tables 3-11 and 3-12, only 351 of the 1000 harshly faulty data points were removed. Before removal, the SMA filter propagated the faults of those data points throughout the dataset. This highlights the importance of ensuring that any dataset contains good data before doing any signal processing. The signal processing can potentially hide the faulty data from the naked eye and could produce harmful results if it is passed to a trained machine learning algorithm attempting to classify important parameters.

3.2.4 Test #4: Test With 1000 Mildly Faulty Data Points in Testing Set
The testing set in Test #4 contains 1000 random data points (the same points as in Test #2 and Test #3) whose HPC and LPT total outlet temperatures are mildly faulty. Half of the 1000 testing examples have their HPC total outlet temperature multiplied by 0.9 and the other half have their LPT total outlet temperature multiplied by 0.8. These constant factors increase the adiabatic efficiency of the compressor and turbine sections of the engines to about 1.05. Figures 3-27 through 3-31 below show the scatter plots of the HPC vs. LPT total outlet temperatures for each of the remaining combinations of SMA filter, feature mean normalization, and multiclass/binary classification.
Figure 3-27: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples

Figure 3-28: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples
Figure 3-29: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples

Figure 3-30: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples
Figures 3-27 through 3-31 show similar trends to their harshly faulty counterparts in Figures 3-17-21. For the unfiltered data in Figures 3-27 and 3-28 the data points have no influence over each other so the data exists in three clusters of only good data, good LPT but bad HPC total outlet temperature data, and good HPC but bad LPT total outlet temperature. The filtered data in Figures 3-29 and 3-30 shows the same sprinkling effect, except this time the bad data does not stray as far away from the good data as it did in the harshly faulty case. This is because the mildly faulty numbers are able to be smoothed back closer to the good data by the SMA filter. Interestingly, the data in Figure 3-31 looks extremely similar to the data in Figure 3-21 even though both datasets were subjected to different degrees of faultiness. Since the SMA filter looks at the mean of a group of data points, multiplying data points by a constant only works to shift the average and standard deviation. Once the data is feature mean normalized, this
shift is hidden. This is important because it implies that, in this situation, an engineer will not be able to tell the degree to which their data is faulty. Given the mild nature of the faults in these testing sets, the results of Test #4 indicate how sensitive these models are to bad data. Table 3-14 on the next page compares the results of Test #4 to the results of Test #1, the test on good data, and Test #2, the test in which 1000 random data points are harshly faulty.
Table 3-14: Summary of Model Performance on Testing Set with 1000 Mildly Faulty Examples Compared to Performance on Testing Set with 1000 Harshly Faulty Examples and Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test #1</th>
<th>1000 Harshly Faulty Test #2</th>
<th>1000 Mildly Faulty Test #4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
<td>0.807</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.817</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
<td>0.736</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
<td>0.763</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
<td>0.823</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
<td>0.825</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
</tr>
</tbody>
</table>
Similarly to the Test #2 results, the F1 scores of each model decrease when asked to make predictions on the new testing set with 100 mildly faulty data points. Further, the Fine Gaussian SVM models again see their performance diminish worse than the Linear SVM models. There is a slight yet noticeable increase in F1 score from Test #2 to Test #4, implying that the less faulty the data is, the better the trained models will perform. This makes sense because the models are all trained on good data, so the closer any faulty testing set data gets to the good data, the better the models will perform. Again, the Fine Gaussian SVM trained on SMA filtered, unnormalized, binary training data does not see the same drop in performance as the other Fine Gaussian SVM models. This further supports the notion that SMA filtered, unnormalized, binary training data is the best data with which to train models with using the C-MAPSS dataset. Regardless, though, it is clear that any degree of faultiness significantly impacts the models’ ability to perform well on new data.

3.2.5 Test #5: Test With 1000 Mildly Faulty Data Points Using Physics-Based Filter

The testing set in Test #5 is ran through the physics-based filter after all the data is made faulty and signal processing is done. Any remaining data points that violate the laws of thermodynamics are removed from the dataset. For the unfiltered testing sets, the physics filter removes all 1000 points that are faulty. For the filtered testing sets, the physics filter does not catch all the faulty data points because most have been smoothed to below the removal threshold due to the good data points surrounding them. It is even harder for the physics filter to remove data points in this testing set because of the mild nature of the faults. The adiabatic efficiencies of both the compressor and turbine sections are only increased to 1.05 as opposed to 1.25 in the harshly faulty testing set. The SMA filter should smooth the vast majority of these points to below detection levels. Tables 3-15 through 3-18 below show the classification distributions of
the remaining examples in the unfiltered and filtered testing sets for both multiclass and binary classification.

Table 3-15: Summary of Classification Distribution of Unfiltered Multiclass Testing Set with 1000 Mildly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1812</td>
<td>14.98%</td>
</tr>
<tr>
<td>Medium</td>
<td>5527</td>
<td>45.69%</td>
</tr>
<tr>
<td>Short</td>
<td>3933</td>
<td>32.51%</td>
</tr>
<tr>
<td>Urgent</td>
<td>824</td>
<td>6.81%</td>
</tr>
</tbody>
</table>

Table 3-16: Summary of Classification Distribution of Unfiltered Binary Testing Set with 1000 Mildly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>11272</td>
<td>93.19%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>824</td>
<td>6.81%</td>
</tr>
</tbody>
</table>

Table 3-17: Summary of Classification Distribution of Filtered Multiclass Testing Set with 1000 Mildly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1956</td>
<td>14.94%</td>
</tr>
<tr>
<td>Medium</td>
<td>5988</td>
<td>45.75%</td>
</tr>
<tr>
<td>Short</td>
<td>4258</td>
<td>32.53%</td>
</tr>
<tr>
<td>Urgent</td>
<td>886</td>
<td>6.77%</td>
</tr>
</tbody>
</table>

Table 3-18: Summary of Classification Distribution of Filtered Binary Testing Set with 1000 Mildly Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>12202</td>
<td>93.23%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>886</td>
<td>6.77%</td>
</tr>
</tbody>
</table>

There are 12096 testing examples in the unfiltered testing sets and 13088 testing examples in the filtered testing sets. This shows just how many data points that the SMA filter is able to smooth under the removal threshold. This time, only 8 of the 1000 faulty data points are removed in the filtered testing sets, compared to 345 removed faulty data points in the harshly
faulty testing sets from Test #3. This further highlights how signal processing can work to hide bad data in a given dataset. Figures 3-32 through 3-36 show the scatter plots of the testing sets that each model is asked to make predictions on.

![Figure 3-32: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter](image)

*Figure 3-32: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter*
Figure 3-33: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter

Figure 3-34: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter
Figure 3-35: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter

Figure 3-36: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with 1000 Mildly Faulty Examples After Being Ran Through the Physics Filter
Figures 3-32 through 3-36 show the effectiveness, or lack thereof, of the physics-based filter on datasets with various levels of signal processing. Similarly with Test #3, the physics filter returns both the multiclass and binary unfiltered testing sets back to their original state, albeit with 1000 less data points. It is impossible to tell that the unfiltered data was ever faulty. In Figures 3-34 and 3-35, the datasets where the SMA filter is applied, there is not much change at all compared to Figures 3-29 and 3-30. The physics filter is only able to remove 8 points from the dataset, showing its ineffectiveness on SMA filtered data that is mildly faulty. Looking at Figure 3-36, the most extreme outliers are removed compared to Figure 3-31. This is evidenced by the minimums of the x and y axes in Figure 3-36 being at -7 and -8, respectively, compared to both being at -8 in Figure 3-31.

Since all bad data points are removed in the unfiltered testing sets, the performance of the modes trained on unfiltered training data is expected to approach the performance of those models on Test #1, the testing set with only good data. The models trained on filtered data, though should not return to their Test #1 performance because faults still exist within the testing set for Test #5. In fact, the performance of the filtered models on Test #5 should closely follow their performance on Test #4 because only 8 testing examples are removed from the testing set. Table 3-19 on the next page compares the results of Test #5 to those of both Test #4 and Test #1.
Table 3-19: Summary of Model Performance on Testing Set with 1000 Mildly Faulty Examples Ran Through Physics Filter Compared to Performance on Testing Set with 1000 Mildly Faulty Examples and Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test #1</th>
<th>1000 Mildly Faulty Test #4</th>
<th>1000 Mildly Faulty Out Test #5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1% 0.611 0.807 0.695</td>
<td>58.1% 0.571 0.811 0.670</td>
<td>58.9% 0.610 0.810 0.696</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4% 0.602 0.817 0.693</td>
<td>96.2% 0.563 0.819 0.667</td>
<td>96.4% 0.602 0.818 0.694</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5% 0.696 0.736 0.715</td>
<td>49.0% 0.710 0.355 0.473</td>
<td>52.5% 0.688 0.733 0.710</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>yes</td>
<td>no</td>
<td>51.9% 0.635 0.763 0.693</td>
<td>36.7% 0.726 0.135 0.228</td>
<td>36.7% 0.726 0.135 0.228</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>yes</td>
<td>no</td>
<td>96.5% 0.623 0.823 0.709</td>
<td>96.5% 0.597 0.845 0.700</td>
<td>96.5% 0.597 0.845 0.700</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>yes</td>
<td>no</td>
<td>60.3% 0.626 0.825 0.712</td>
<td>58.1% 0.547 0.860 0.669</td>
<td>58.1% 0.547 0.860 0.669</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>yes</td>
<td>yes</td>
<td>93.4% 0.885 0.509 0.646</td>
<td>92.2% 0.489 0.432 0.459</td>
<td>92.2% 0.489 0.432 0.459</td>
</tr>
</tbody>
</table>
As expected, the precision, recall, and F1 scores on Test #5 of the models trained and tested on unfiltered data returned to those of Test #1. This is the same result as Test #3 because the testing sets in Test #5 and Test #3 are exactly the same since all faulty data points are removed. Also as expected, the precision, recall, and F1 scores on Test #5 of the models trained and tested on filtered data remained about the same as compared to Test #4. This is because only 8 testing examples are removed from the testing set, so there is not much room for a noticeable performance change. The SMA filter propagates the faults of all 1000 mildly faulty data points throughout the dataset. This further emphasizes the importance of ensuring that any dataset contains good data before applying any signal processing since all trained models perform poorly regardless of whether this fault is harsh or mild. In the face of faulty data, though, the Fine Gaussian SVM trained on SMA filtered, unnormalized, binary training data is able to make consistent predictions on all testing sets which makes it a great candidate for future optimization and potential retraining with more training examples. This idea is discussed further in the Future Work section of the Conclusion chapter. This concludes all tests where 1000 random data points are faulty.

3.2.6 Test #6: Test With Even-Numbered Engine Harsh Faults In Testing Set
The testing set in Test #6 contains 50 engines whose HPC or LPT total outlet temperature data is harshly faulty. This test case represents the scenario where an improper and/or uncalibrated sensor is placed on an engine for data acquisition. An example that was previously discussed is if a K type thermocouple is used to measure a temperature of interest when a T type thermocouple is more suitable. In the testing sets for Test #6, engines 2, 6, 10, 14 and so on up to 98 have their HPC total outlet temperatures multiplied by 0.8. Engines 4, 8, 12 and so on up to 100 have their LPT total outlet temperatures multiplied by 0.5. Just as with the previous testing sets, this harsh fault increases the adiabatic efficiency of the compressor and turbine
sections of the engines to about 1.25. Figures 3-37 through 3-41 below show the scatter plots of the HPC vs. LPT total outlet temperatures for each of the remaining combinations of SMA filter, feature mean normalization, and multiclass/binary classification.

![Figure 3-37: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Harsh Faults on Even-numbered Engines](image)

Figure 3-37: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Harsh Faults on Even-numbered Engines
Figure 3-38: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Harsh Faults on Even-numbered Engines

Figure 3-39: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Harsh Faults on Even-numbered Engines
Figure 3-40: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Harsh Faults on Even-numbered Engines

Figure 3-41: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Harsh Faults on Even-numbered Engines
Upon inspection of Figures 3-37 through 3-41, the unfiltered and filtered unnormalized testing sets are strikingly similar. This is in stark contrast to the previous test cases in which the SMA filter caused the faulty data to propagate through the filtered testing set. In this test case, since the faults are applied to the entirety of available data for a given engine, the faults do not propagate to the rest of the idea. The SMA filter is applied engine by engine so that slight differences in sensor values from engine to engine do not affect the entire dataset. As such, for the engines with faulty data, the SMA filter has no good data with which to smooth the faulty data. This causes the clustering effect to be seen on both the unfiltered and filtered unnormalized testing sets. The normalized testing set in Figure 3-41, though, does not appear to change at all compared to Figure 3-16 for Test #1, the testing set that only contained good data. This is, in fact, changing the data for entire engines by a constant factor does not change how the data looks once it is normalized. Since the fault is a constant factor, the average is shifted. When feature mean normalization is applied, this average shift is hidden. Further, since the SMA filter is applied engine by engine, the faulty data is only being filtered with the same faulty data. This order of signal processing makes it impossible to tell whether there is any faulty data in the testing set without the physics-based filter. Table 3-20 on the next page compares the results of Test #6 to the results of Test #1.
### Table 3-20: Summary of Model Performance on Testing Set with Even Engine Harsh Faults Compared to Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test #1</th>
<th>Even Engine Harsh Fault Test #6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
</tr>
</tbody>
</table>
The most obvious takeaway from Table 3-20 is that the performance of the Fine Gaussian SVM trained on the filtered, normalized, binary training set is unchanged from Test #1 to Test #6. This is expected as the datasets are the exact same due to the masking effect the feature mean normalization has on the faults. While it may seem like a good thing that the performance is maintained even with faulty data, this is not good because it could be very harmful if any bad data is hidden. Engineers need to know if they are getting bad data so that they can determine a solution to fix the problem. The performance of the rest of the models in Table 3-20 is decreased across the board, moreso than in previous test cases where 1000 data points are faulty. This makes sense because not only are more than 1000 data points faulty (half the engines make up about half of the testing examples in the testing set), but there is no smoothing effect to bring some of those points back closer to their regular range.

3.2.7 Test #7: Test With Even-Numbered Engine Harsh Faults Using Physics-Based Filter
The testing set in Test #7 is ran through the physics-based filter after all the data is made faulty and signal processing is done. Any remaining data points that violate the laws of thermodynamics are removed from the dataset. Since the SMA filter does not help to smooth the faulty data in the filtered testing sets, the physics filter removes all faulty testing examples in both the SMA unfiltered and filtered testing sets. Ultimately, this yields the same testing sets used in Test #1, except with only the odd engines’ data present in the testing set. Tables 3-21 and 3-22 below show the classification distributions of the remaining examples for both multiclass and binary classification. Due to the way the SMA filter is applied to the filtered testing sets, the distribution is the same for the filtered and unfiltered testing sets.
Table 3-21: Summary of Classification Distribution of Multiclass Testing Set with Even Engine Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>1148</td>
<td>18.36%</td>
</tr>
<tr>
<td>Medium</td>
<td>3049</td>
<td>48.76%</td>
</tr>
<tr>
<td>Short</td>
<td>1751</td>
<td>28.00%</td>
</tr>
<tr>
<td>Urgent</td>
<td>305</td>
<td>4.88%</td>
</tr>
</tbody>
</table>

Table 3-22: Summary of Classification Distribution of Binary Testing Set with Even Engine Faulty Data Points Removed

<table>
<thead>
<tr>
<th>Classification</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>5948</td>
<td>95.12%</td>
</tr>
<tr>
<td>Unhealthy</td>
<td>305</td>
<td>4.88%</td>
</tr>
</tbody>
</table>

With the even engine data entirely removed from the testing sets, there are now 6253 testing examples remaining compared to the 13096 examples in the original testing set. This is slightly less than half of the total examples in the testing set, and the distribution of the classes has slightly changed as well. Originally there were 886 urgent/unhealthy testing examples and that number is now down to 305, a 65% decrease. Figures 3-42 through 3-46 below show the scatter plots of the HPC vs. LPT total outlet temperatures for each of the remaining combinations of SMA filter, feature mean normalization, and multiclass/binary classification.
Figure 3-42: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter

Figure 3-43: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter
Figure 3-44: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter

Figure 3-45: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter
Figure 3-46: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Even Engine Harsh Faults After Being Ran Through the Physics Filter

The most important observation to make about Figures 3-42 through 3-46 is that they have the exact same data as Figures 3-9, 3-10, 3-13, 3-14, and 3-16 except that just over half of the data points are removed by the physics filter. Each trained model is expected to perform just as well on this testing set as it did on the testing set for Test #1. The exact values of recall, precision, and F1 score may be slightly different though, because of the removal of 65% if the urgent/unhealthy examples. Ultimately, though, each testing set now only contains good data that is completely unaffected by the faulty data that was in the testing sets before the physics filter removed it. Table 3-23 on the next page compares the of Test #7 to the results of Test #6 and Test #1, the test with the full dataset containing only good data.
Table 3-23: Summary of Model Performance on Testing Set with Even Engine Harsh Faults Ran Through Physics Filter Compared with Performance on Testing Set with Even Engine Harsh Faults and Testing Set with Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test #1</th>
<th>Even Engine Harsh Faults Test #6</th>
<th>Even Engine Harsh Faults Out Test #7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
<td>0.807</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.817</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
<td>0.736</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
<td>0.763</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
<td>0.823</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
<td>0.825</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
</tr>
</tbody>
</table>
As expected, the recall, precision, and F1 scores on Test #7 of each of the models approach their counterparts on Test #1. Interestingly, though, none of the models actually return to their exact initial recall, precision, and F1 scores. This is because the odd engines remaining in these testing sets are harder to make predictions on than the even units with good data. Since nothing else has changed about the Test #7 testing set compared to the Test #1 testing set, the reason for the performance drop has to be because the odd numbered engines behave slightly differently than the even numbered engines. This observation is purely coincidental and is not an artifact inserted by the creators of the C-MAPSS dataset. This is best evidenced by the fact that the Fine Gaussian SVM trained on SMA filtered, feature mean normalized, binary training data sees a drop in F1 score on Test #7 compared to Test #6 and Test #1. Recall that the feature mean normalization masks the data faults so that the testing sets in Test #6 and Test #1 are the exact same. As such, the performance drop in Test #7 is chalked up to the odd engines’ good data being harder to make predictions on. The results of Test #7 highlight the importance of ensuring that the sensors wired to any engine are, in fact, suitable to take the desired measurements. Further, it is also important to ensure that all components along the data transmission path from sensor to DAQ computer are functioning properly. Even if only 2 of 14 sensors used to train a machine learning model start giving bad data for an engine, the model is no longer able to maintain its performance on new data.

3.2.8 Test #8: Test With Even-Numbered Engine Mild Faults In Testing Set
The testing set in Test #8 contains 50 engines whose HPC or LPT total outlet temperature data is mildly faulty. This test case represents the same scenario as Test #6, with a lessened degree of faultiness. In the testing sets for Test #8, engines 2, 6, 10, 14 and so on up to 98 have their HPC total outlet temperatures multiplied by 0.9. Engines 4, 8, 12, and so on up to 100 have their LPT total outlet temperatures multiplied by 0.8. This mild fault increases the adiabatic
efficiency of the compressor and turbine sections of the engines to about 1.05. These are the same mild fault factors used to add faults to 1000 random data points in Test #4. Figures 3-47 through 3-51 below show the scatter plots of the HPC vs. LPT total outlet temperatures for each of the remaining combinations of SMA filter, feature mean normalization, and multiclass/binary classification.

Figure 3-47: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Multiclass Testing Set with Mild Faults on Even-numbered Engines
Figure 3-48: Scatter Plot of HPC and LPT Outlet Temperatures for the Unfiltered, Unnormalized, Binary Testing Set with Mild Faults on Even-numbered Engines

Figure 3-49: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Multiclass Testing Set with Mild Faults on Even-numbered Engines
Figure 3-50: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Unnormalized, Binary Testing Set with Mild Faults on Even-numbered Engines

Figure 3-51: Scatter Plot of HPC and LPT Outlet Temperatures for the Filtered, Normalized, Binary Testing Set with Mild Faults on Even-numbered Engines
Figures 3-47 through 3-51 are very similar to their harsh fault counterparts in Figures 3-37 through 3-41. The only exception here is that the faults are mild, so the x and y axes have tighter scales compared to the scatter plots showing harsh faults. Further Figure 3-51 shows the exact same data as Figure 3-41 (harsh faults) and Figure 3-16 (good data). Again, this is because the engine-by-engine feature mean normalization hides the data faults and since the SMA filter is also applied engine-by-engine, there is no good data with which to smooth the faults. Given the mild nature of the faults in these testing sets, the results of Test #8 indicate how sensitive these models are to bad data. Table 3-24 on the next page compares the results of Test #8 to the results of Test #1, the test on good data, and Test #6, the test in which even engine HPC and LPT outlet temperature data is harshly faulty.
Table 3-24: Summary of Model Performance on Testing Set with Even Engine Mild Faults Exam Compared to Performance on Testing Set with Even Engine Harsh Faults and Performance on Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Good Data Test #1</th>
<th>Even Engine Harsh Faults Test #6</th>
<th>Even Engine Mild Faults Test #8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
<td>0.807</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.817</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
<td>0.736</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
<td>0.763</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
<td>0.823</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
<td>0.825</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
</tr>
</tbody>
</table>
Table 3-24 shows that all models either exhibit either the same or improved performance when asked to make predictions on testing sets with even engines that have mildly faulty HPC and LPT total outlet temperatures. For the Fine Gaussian SVM model trained on SMA filtered, normalized, binary training data, the performance on Test #8 is the same as on Test #6 and Test #1 because the same exact testing sets are used for those cases. There are three other models whose recall, precision, and F1 score do not change whether the fault is harsh or mild. These are the Fine Gaussian SVM model trained on the unfiltered, unnormalized, multiclass training set, the Fine Gaussian SVM model trained on the SMA filtered, unnormalized, multiclass training set, and the Linear SVM model trained on the SMA filtered, unnormalized, multiclass training set. This stagnation in performance implies that even the mild fault is enough to bring the examples in the testing set beyond the threshold of faults that the models can handle. This further emphasizes the need to ensure that suitable sensors are used and that there are no faulty components along the path of data transmission from the sensor to the DAQ computer. The Fine Gaussian SVM model trained on SMA filtered, unnormalized, binary training data again shows the most improved performance when going from harsh to mild faults. This model’s ability to make some reasonable (compared to the other models) predictions on mildly faulty data makes it’s training characteristics desirable for further studies; these may be the best combination of signal processing techniques to make classification predictions on the C-MAPSS dataset.

3.2.9 Test #9: Test With Even-Numbered Engine Mild Faults Using Physics-Based Filter
The testing set in Test #9 is ran through the physics-based filter after all the data is made faulty and signal processing is done. Any remaining data points that violate the laws of thermodynamics are removed from the dataset. This yields the same exact testing sets as in Test #7 for all combinations of SMA filter, feature mean normalization, and binary/multiclass classification. For that reason, scatter plots are omitted because they are the exact same as for
Test #7. Refer to Figures 62-66 to see the scatter plots. Even though Test #9 gives the same results as Test #7, it is convenient to see these results compared to those of Test #8 and Test #1, the test on good data. Table 3-25 on the next page shows these results.
Table 3-25: Summary of Model Performance on Testing Set with Even Engine Mild Faults Ran Through Physics Filter Compared with Performance on Testing Set with Even Engine Mild Faults and Testing Set with Good Data

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Classification Method</th>
<th>Engine List</th>
<th>SMA Filter</th>
<th>Feature Mean Normalization</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>59.1%</td>
<td>0.611</td>
<td>0.807</td>
<td>0.695</td>
<td>50.2%</td>
<td>0.227</td>
<td>0.696</td>
<td>0.342</td>
<td>55.8%</td>
<td>0.597</td>
<td>0.674</td>
<td>0.633</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Binary</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>96.4%</td>
<td>0.602</td>
<td>0.817</td>
<td>0.693</td>
<td>94.1%</td>
<td>0.222</td>
<td>0.704</td>
<td>0.338</td>
<td>96.7%</td>
<td>0.593</td>
<td>0.686</td>
<td>0.636</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>no</td>
<td>no</td>
<td>52.5%</td>
<td>0.696</td>
<td>0.736</td>
<td>0.715</td>
<td>27.5%</td>
<td>0.872</td>
<td>0.108</td>
<td>0.192</td>
<td>48.3%</td>
<td>0.630</td>
<td>0.610</td>
<td>0.620</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>51.9%</td>
<td>0.635</td>
<td>0.763</td>
<td>0.693</td>
<td>27.4%</td>
<td>0.853</td>
<td>0.106</td>
<td>0.189</td>
<td>48.2%</td>
<td>0.574</td>
<td>0.608</td>
<td>0.591</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>96.5%</td>
<td>0.623</td>
<td>0.823</td>
<td>0.709</td>
<td>94.9%</td>
<td>0.336</td>
<td>0.780</td>
<td>0.470</td>
<td>96.7%</td>
<td>0.597</td>
<td>0.684</td>
<td>0.638</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>1v1</td>
<td>All</td>
<td>Yes</td>
<td>no</td>
<td>60.3%</td>
<td>0.626</td>
<td>0.825</td>
<td>0.712</td>
<td>50.0%</td>
<td>0.205</td>
<td>0.687</td>
<td>0.316</td>
<td>56.9%</td>
<td>0.597</td>
<td>0.687</td>
<td>0.639</td>
</tr>
<tr>
<td>Fine Gaussian SVM</td>
<td>Binary</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
<td>0.646</td>
<td>93.4%</td>
<td>0.885</td>
<td>0.509</td>
<td>0.646</td>
<td>94.0%</td>
<td>0.925</td>
<td>0.445</td>
<td>0.601</td>
</tr>
</tbody>
</table>
Ultimately, the same can be said about the results of Test #9 in Table 3-25 as was said for the results of Test #7 in Table 3-23 (harshly faulty data out table). The recall, precision, and F1 score of all models approach their respective scores on Test #1, the test with good data. Again, the reason for the slightly diminished performance on Test #9 is due to the pure coincidence that the even-numbered engines in these testing sets are harder to make classification predictions on then the odd-numbered engines.

Overall, the efficacy of the physics-based filter is displayed through these results. While the physics-based filter is unable to correct bad data so that useful predictions can be made on it, it is able to identify bad data and, in a real-setting, can alert an engineer that there is a data problem. Further, applying the physics-based filter before any signal processing is done mitigates any prediction issues caused by the bad data. The bad data is identified right away, removed from the testing sets, and useful predictions are made on the remaining data.
4 Conclusions
4.1 Summary
This thesis uses a complete simulated gas turbine engine run-to-failure dataset created in the C-MAPSS engine simulation software package [30] to train Support Vector Machine machine learning models [21] and test them on faulty data. The faulty data aims to emulate two realistic scenarios that engineers may encounter when attempting to collect data from a gas turbine engine. The HPC and LPT total outlet temperatures were the sensor outputs of interest that are faulty. One scenario is that of a degraded/faulty sensor and/or a degraded/faulty component along the path of data transmission from the sensor to the DAQ computer. In this scenario, sensor data may be intermittently faulty, which alters some data points but leaves others untouched. To test this scenario, both a harsh and mild data fault scheme is implemented for 1000 random data points in the testing set. The harsh scheme intended to add faults to these temperature measurements such that the adiabatic efficiency of the compressor and turbine sections increases to around 125%. The mild scheme intended to bring the adiabatic efficiencies to around 105%; both schemes make it so that the data violate the laws of thermodynamics. The second scenario is that of an uncalibrated/unsuitable sensor being used to collect data. In this scenario, the data for the engine equipped with this sensor could consistently be faulty by a constant factor. To test this scenario, the same harsh and mild schemes were implemented except, rather than 1000 random points, on the entire column of HPC and LPT total outlet temperatures for even-numbered engines.

The SVM machine learning models are trained on C-MAPSS training sets prepared with various levels of signal processing techniques including the simple moving average (SMA) filter, feature mean normalization, and binary/multiclass classification. The models are trained using Matlab’s Classification Learner application. Both Gaussian and Linear SVM models are trained.
Each model is asked to predict the health of a gas turbine engine based on its sensor outputs for a given flight. For multiclass classification, the models are asked to classify points as long, medium, short, or urgent need of maintenance. For binary classification, the models are asked to classify points as healthy or unhealthy. All trained models are evaluated on their recall, precision, and F1 score corresponding to their ability to classify the urgent/unhealthy class correctly. The models that performed best on their training sets moved on to be tested on testing sets contained faulty data where the same performance benchmarks are used.

A novel physics-based filter is introduced to curb the potentially harmful effects of each of these scenarios. Datasets are fed through the physics-based filter to remove any data points that correspond to isentropic efficiencies of the compressor or turbine section being greater than 100%. The discussed signal processing techniques effect the way that faulty data are/are not propagated through the testing sets and thus how the trained models perform on these testing sets. Through applying the physics-based filter to these testing sets and testing the models, the important of the order of signal processing techniques is illuminated. When signal processing such as SMA filtering and feature mean normalization is done before applying the physics-based filter, the faulty data can propagate through the dataset render it useless for making useful predictions. When the physics-based filter is applied before any other signal processing is done, it is able to catch faulty data before it has a chance to affect the rest of the dataset. This alerts the engineer responsible for monitoring the data that there is a data collection problem, while also allowing useful predictions to be made on the remaining good data in the dataset.

4.2 Future Work

There are a few interesting paths that can be taken for future studies given the conclusions of this thesis. One path is to train additional models on training sets that do not
include HPC and LPT total outlet temperature as a feature. These would serve as backup models that can be used to make predictions when the physics-based filter identifies an issue with the HPC and/or LPT total outlet temperature data. These models would eliminate the need for that flight’s entire sensor output to be remove from the dataset. Instead, one can simply remove the faulty data, i.e. HPC and/or LPT total outlet temperature, and use the remaining data with the backup model to make predictions. Another path is to implement a method to fix the faulty data rather than remove it entirely. This may be especially useful for the data acquisition scenario of a degraded/faulty sensor and/or transmission path component. Instead of removing the data, it can be replaced with a reasonable value determined by the previous good sensor values. A third future study could involve finding/creating more training examples with which to retrain the well-performing models from this thesis with. All trained models exhibited high variance, so additional training examples would improve their ability to generalize to new examples.
5 References


