Photocopy and Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

Auto Machine Learning Applications for Nuclear Reactors: Transient Identification, Model Redundancy and Security

by

Pedro Mena

pepomena@isu.edu

Department of Computer Science

Idaho State University

Spring 2022

A dissertation submitted to partially fulfill the requirements for the degree of Doctor of Philosophy in Applied Science and Engineering at Idaho State University.

Committee Approval

To the Graduate Faculty:

The members of the committee appointed to examine the dissertation of Pedro Mena find it satisfactory and recommend that it be accepted.

Dr. Leslie Kerby
Committee Chair

Dr. R.A Borrelli
Committee Member

Dr. Paul Bodily
Committee Member

Dr. Xiaoxia Xie
Committee Member

Dr. Donna Delparte
Graduate Faculty Representative

Dedication

I would like to dedicate this dissertation to my family: my mother Rita, my father Arturo and my siblings Arturo and Selenia. This work would not have been possible without their constant support and encouragement.

Acknowledgement

This research made use of Idaho National Laboratory computing resources, which are supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517.

Special thanks to John Petersen for support with the GPWR simulator, Stephanie J. Parker for support with INLHPC, David Rodgers for support with funding and John A. Koudelka for support with the CAES Audio Visual Lab.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network. 6, 146, 147

BOL Beginning of Life. 68, 70–72, 75, 96, 97

BPN Back Propagation Network. 16

BWR Boiling Water Reactor. 10

CAES Center for Advance Energy Studies. 61

CBM Condition Based Maintenance. 7

CFR Code of Federal Regulations. 10, 11

CNN Convolutional Neural Network. 13, 16, 147, 148, 168

CPU Central Processing Unit. 2, 148

CSV Comma Separated Values. 71, 74

DOE Department of Energy. 15, 145, 161, 183

EBR-2 Experimental Breeder Reactor 2. 15

ECCS Emergency Core Cooling System. 9

EOL End of Life. 68–72, 75, 96, 97

FWER Family Wise Error Feature Rate Selection. 45–47

G.E General Electric. 10, 14

GAN Generative Adversarial Network. 184, 185

GAO Government Accountability Office. 145

GPU Graphical Processing Unit. 2, 28, 148, 164, 190

GPWR Generic Pressurized Water Reactor. iv, xii, 61, 66–68, 95–97, 143, 150, 159, 169, 178, 179, 185–189

GUI Graphical User Interface. 163, 165, 169, 170, 175, 176, 180, 190

HPC High Performance Computing. 101, 129, 131, 150, 164, 169, 174, 185

HRA Human Reliability Assessment. 183

IAEA International Atomic Energy Agency. 10, 11, 61

INL Idaho National Lab. 11, 15, 20, 61, 66, 101, 169, 183

kNN k-nearest neighbors. x, xii, 6, 116–119, 121, 124, 128, 130, 132, 167, 186, 187

LOCA Loss of Coolant Accident. 9

LOFA Loss of Flow Accident. 15, 17, 18

LOOP Lose of Off-Site Power. 71

MARS Multi-dimensional Analysis of Reactor Safety. 17

MOL Middle of Life. 68, 70–72, 75, 96, 97

MSIV Main Steam Isolation Valves. 70, 72

NAMS Nearly Autonomous Management System. 15, 16

NCSU North Carolina State University. 15, 16, 18, 66

NRC Nuclear Regulatory Commission. 10, 11, 17, 18, 66, 69, 77, 93, 145, 161, 181–183

ORNL Oak Ridge National Lab. 13, 14

PCA Principal Component Analysis. 44, 56, 147

PdM Predictive Maintenance. 7

PORV Pilot Operating Relief Valve. 8, 9, 136, 161

PRA Probabilistic Risk Assessment. 181, 183

PvM Preventive Maintenance. 7

PWR Pressurized Water Reactor. 14, 16

R2F Run to Failure. 7

RBF Radial Basis Function. 42

RCS Reactor Coolant System. 75, 137, 139

RNN Recurrent Neural Network. 16, 148

SMR Small Modular Reactor. 183

SVM Support Vector Machines. 6, 17, 167, 184, 185

SVR Support Vector Regression. 14

TMI Three Mile Island. 8–10, 15, 66, 103, 161

TPOT Tree-based Pipeline Optimization Tool. xii, 27–30, 32, 35, 45, 48, 68, 78, 91, 101, 103–105, 119, 122, 128–133, 136, 138, 143, 144, 163–167, 169–171, 173–175, 178, 179, 186–190

V&V Verify & Validate. 11, 66

WSC Western Services Cooperation. 61

Auto Machine Learning Applications for Nuclear Reactors: Transient Identification, Model Redundancy and Security

Dissertation Abstract – Idaho State University (2022)

Machine learning and AI are concepts that have had a large impact in daily life since 2000. It is unlikely that most people at this point in time do not have some sort of interaction with an AI system on a daily basis. This research effort looked to contribute to the field of nuclear safety and explore ways to expand the use of machine learning through the application of AutoML. This project consisted of four major phases. In the first phase, data was collected from a GPWR simulator for five different reactor events, creating a dataset with over 30,000 points. Six different machine learning models were trained using the AutoML package TPOT. The results from this test were positive with all models producing accuracies in the high 90% range. The models were also able to perfectly distinguish a reactor operating normally from one experiencing a transient. In the next phase, the dataset was expanded using the GPWR, the number of classes was increased to 12 and the new dataset consisted of over 110,000 points. Models were retrained and while many of models suffered in validation, three of the models were still able to score results in the low 90% range. The models were then examined looking at model redundancy by dropping key features, examine variation due to changes in random state, exploring ways to improve the model and identify the reasons behind misclassifications. The third phase of the project explored the use of autoencoders to identify GPWR data that had been altered. The model was able to identify all points at high levels of noise, but performance dropped off as the noise was decreased. Still, the technique has validity to help with security concerns and identify sensor malfunctions. The final phase of the project was to explore different AutoML approaches and compare and contrast their performance, ease of use and functionality. These were TPOT, H2O and Google Cloud AutoML. Each of these approaches were found to have different advantages and issues, but all performed with models produced using GPWR data, with results in the mid to high 90% range.

Keywords: Machine Learning, Nuclear Safety, AutoML, Anomaly Detection, Nuclear Simulation, Data Science

1 Introduction

The rapid development of computer technology and the widespread use of the internet has led to exponential growth in the area of data science and artificial intelligence. The ability to collect, store and process large amounts of data more easily has led to a mass adaptation of data science in several industries. Today, researchers and businesses are exploring new applications for this technology in order to improve efficiencies and add value to operations. The purpose of this dissertation is to explore the use of Auto Machine Learning or AutoML in the area of reactor transient diagnosis, as well as to contribute to the effort to implement machine learning to the field of reactor safety.

1.1 Background

This section will provide an overview of the concept of machine learning as well as the motivations and benefits of adapting artificial intelligence in today's environment. This will include a review of the well known applications of machine learning and data science across all industries including safety, business, operations, etc. This section will also include an overview of the history of nuclear safety and the events that shape how the field has grown and developed to its current state. A comprehensive review of the proposed uses of machine learning in several areas within the nuclear safety field will be performed. Finally, proposed approaches for identifying reactor transients will be examined.

1.1.1 Machine Learning

Machine learning is an area in computer science used to make predictions. The interest in the concept of machine learning has grown exponentially in recent times. Once considered too computationally expensive, machine learning has become a part of most people's lives, without most people realizing it. This is in part due to advancements in

computational capabilities in the areas of multi-core Central Processing Unit (CPU)s and Graphical Processing Unit (GPU)s. This has allowed data scientist to develop complex models to make predictions based on provided data. The advantage that machine learning based models have over traditional programming is that these models have the capability to learn over time. Static programs while having the ability to respond to an input, can only respond based on what has been programmed. Machine learning models however can identify trends in the data over time and make decisions based on these trends without having to be reprogrammed.

Machine learning models have the ability to perform one of two different functions. The first of these is the ability to predict a numeric value using a regression model based on the features of the data that has been provided. Examples of this would be the price of a home, the number of units sold, etc. The second type of prediction a machine learning model can make is referred to as classification. These models use the data provided to determine what category or class a sample belongs to. Classification examples include predicting the specific type of flower from physical characteristics of a sample and determining the category of risk of a proposed loan. This project will focus around classification models.

There are several different types of machine learning models. The two most common types of models are supervised learning and unsupervised learning models. Supervised learning models use data that has an outcome already associated with it. These outcomes can be binary, on/off, exist/doesn't exist ,etc. or multi-class, such as excellent, good, average and poor. In classification models, these outcomes serve as the categories that will be predicted by the model. The advantage of supervised models is that the user can specify the number of categories to be used in the model. It should be noted that supervised models have an increased probability of becoming overfit and the models typically are more expensive computationally. Examples of supervised learning models include decision trees and logistic regression.

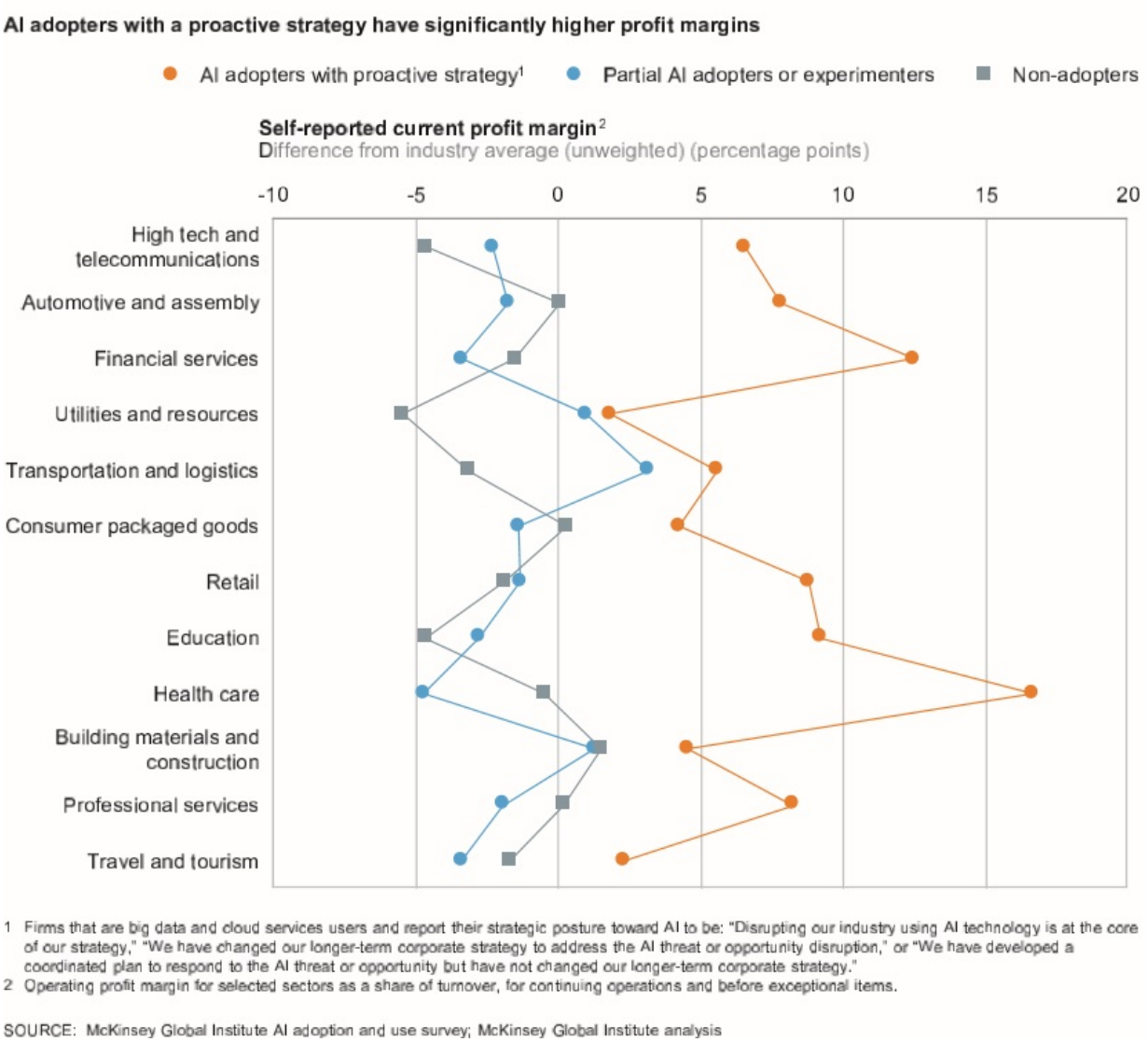
Unsupervised models use data that has no known outcome associated with it. Instead, the model attempts to define groups based on the data provided. These techniques are particularly useful when the data has no set groups. This is especially useful in the area of anomaly detection[1]. Examples of this type of model include clustering techniques, like k-means. It should be noted there are other types of models, such as semi-supervised learning and reinforcement learning, but this project will focus exclusively on supervised and unsupervised methods.

1.1.2 Motivation for Machine Learning Adaptation

Perhaps the largest motivation for the expansion in data science over the last twenty years, is the economic benefit projected by implementing data based approaches. One study from the U.K based company PWC found that the gross domestic product of the United Kingdom could increase by over 10% through the adaptation of artificial intelligence[2]. It should be noted, this study claims most of that projected growth, 8.4%, will be from new ventures that result from the adaptation, while the remaining growth will be a result of productivity improvements.

The magnitude of the economic benefits from the adaptation of artificial intelligence is dependent on a number of factors, including the industry and the approach of adaptation. A study performed by the McKinley Global Institute found that several industries could see improvements with a full adaptation, however in some cases, only a partial adaptation could negatively impact margins[3]. According to this study, industries such as utilities, that historically have low or negative profit margins, due to the regulatory nature of these areas, could see positive margins even with only a partial adaptation of artificial intelligence. This is especially encouraging as one issue surrounding nuclear power has been costs. Other industries identified by this study that could see improved profit margins that relate to the nuclear field are construction, logistics and healthcare. Figure 1 shows a summary graph from this study.

Figure 1: Analysis of the Affect of A.I on Profit Margins by Industry[3]



1.1.3 Applications of Machine Learning Today

Machine learning plays a part in most people's daily lives. One of the most common application of machine learning models is the development of junk email detection. Generally, these models are trained to detect words or phrases that are associated with junk emails. For example, a model could learn that emails with the phrase "you have won" are usually considered junk and will filter these out from the main inbox. The model can continue to develop by learning what emails are considered junk when the user designates an email as such. Another application of machine learning that people use on a daily basis includes personal assistants, such as Apple's Siri and Amazon's Alexa. These tools can learn trends of the user, based on the data that is provided, such as scheduling, and preferences [4].

1.1.3.1 Applications in Business Business and marketing are areas that have been able to take advantage of machine learning and artificial intelligence to improve their operations. Coca-Cola makes use of these techniques to gather data on what drink combinations may interest customers. The company uses soda machines that allow for users to customize their soft drinks to gather data on what customers like. The model is then able to use this data to learn what drink combinations are popular and use that information to help decide what new products to offer at retail [5]. Other uses of machine learning in marketing include the use of recommendations to offer suggestions to customers. In these cases, the suggestions are generated based on the data collected from the users. For example, Amazon's recommendation algorithm examines the items purchased, rather than the users themselves. This allows for a less data intensive algorithm that has been considered far more effective [6]. Many companies use similar machine learning algorithms to make recommendations to customers, including Netflix, Hulu, Booking.com and Spotify.

Studies have found that machine learning is an effective approach to detecting credit

card and financial fraud. One study used data from over 250,000 European credit card transactions. Three models were created using different machine learning models and two of these models were able to identify fraudulent transactions over 97% of the time[7]. Financial institutions have embraced the use of machine learning models to improve their business. For example,JP Morgan uses a variety of classification techniques to evaluate different areas of the banks operations [8].

1.1.3.2 Applications in Healthcare Due to the high potential for improved profit margins and the ever changing landscape of the industry, healthcare has been quick to adapt machine learning and data science. Providers are looking at ways machine learning can improve the process of storing and sharing patient data between facilities. Image recognition algorithms are being used to help identify areas of interest with screening images. One example of the impact of machine learning in healthcare is at Regional Cancer Care Associates in New Jersey. Due to the COVID-19 pandemic, the facility had experienced issues with staffing and identifying patients who are at high risk for the virus. Machine learning models were used to attempt to classify patients based on the degree of risk [9]. Early results have been positive with the facility reporting improvement in both, identifying high risk individuals and managing the facility.

1.1.3.3 Applications in Diagnostics & Maintenance An application that has been gaining increased use and interest is the use of machine learning models to diagnose maintenance issues with equipment. Since 2000, many organizations and governments have begun developing and implementing standards for maintenance using computer based models, such as artificial intelligence. These includes IEEE, ISO, IEC, as well as the governments of Germany and China[7]. Common techniques used for diagnostics include decision tress, Artificial Neural Network (ANN), k-nearest neighbors (kNN),Support Vector Machines (SVM),etc.

Studies in predictive maintenance using machine learning have been done in man-

ufacturing and other sectors. The implementation of big data, machine learning, etc. in the manufacturing industry has been referred to as Industry 4.0. There are currently four categories for the use of AI in maintenance: Run to Failure (R2F), Preventive Maintenance (PvM), Condition Based Maintenance (CBM) and Predictive Maintenance (PdM)[10]. Each of these different categories has different uses and applications.

1.1.3.4 Applications in Quality Assurance One of the common uses of machine learning in manufacturing is the use of models to help with quality assurance. In a demonstration performed by IBM, Watson, IBM's artificial intelligence system, analyzed several different parts to look for defects. For example, Watson was able to detect a bent connector pin in a component. IBM has stated that they believe that Watson is able to continuously meet a 92% accuracy, comparable to that of a human inspector. Other applications for this use of Watson is for the inspection of part installation and inspection [11].

A study performed by IBM's T.J. Watson Research Center attempted to use machine learning models to improve the reliability of rail lines[12]. The goal of this study was to reduce the probability of a service disruption by identifying components that require repairs prior to a failure occurring. The hope is that rail companies can find savings in reduced delays occurring to unavailable trains, derailments, etc. Several different approaches have been tried and research continues to develop better models using IBM Watson.

In a study from Zhejiang Sci-Tech University in China, machine learning models were used in an attempt to identify failures in air conditioning and refrigeration equipment. According to the study, one issue in trying to develop a model for this type of equipment is a lack of large quantities of data[13]. In order to address this issue, a Sparse autoencoder, neural network based model was used. Models were trained with a different number of layers to see the impact on validation. Data based on 8 different conditions for the equipment was used to train the model. The accuracy of these models was found to be

high with results in the high 90% range. Other validation measurements, such as precision and recall, were found to be in the mid 80% range for diagnosing a normal operating system and increased for identifying specific failures.

1.1.4 Nuclear Safety

In the nuclear industry, no other area takes a higher priority than the field of safety. The most pressing reason for this is the potential for long lasting consequences in the event of a catastrophic failure. Studies have taken place to improve almost every aspect of a nuclear power plant. These include, accident prevention, quality assurance of components, human factors, plant responses, security, mitigation, etc. This has led to a strict regulation system that has earned the industry a reputation as one of the safest in the world.

Despite this reputation and the high redundancy of nuclear power plant safety system, it is important that the industry embraces the concept of continuous improvement. The industry must always seek to improve safety when possible and most importantly, must never fall in to a sense that nuclear facilities are safe enough. This false sense of security was a contributing factor in the three worst accidents in nuclear history, Fukushima-Daiich in 2011, Chernobyl in 1986 and Three Mile Island (TMI) in 1979.

1.1.4.1 Three Mile Island Accident In the United States, no nuclear related accident has impacted the industry to the extent that the accident with TMI Nuclear Station in Middleton, Pennsylvania has had.This accident occur on the morning of March 28th, 1979 when TMI unit-2 experienced a loss of flow transient event [14]. This resulted in the unit-2 tripping, the insertion of control rods to stop the reactor, as designed. In response to this event, the Pilot Operating Relief Valve (PORV) opened to release pressure from the reactor. Once the pressure reached 15.21 MPa the PORV was supposed to shut automatically, however, a malfunction caused the valve to remain open. This allowed

coolant to flow out of the reactor resulting in a small Loss of Coolant Accident (LOCA) event. To compound this problem, the instruments indicated to the operators that the PORV was closed and other instruments that could have alerted operators to the problem were not easily visible. At this point, coolant was still flowing into the reactor via the Emergency Core Cooling System (ECCS) and reactor coolant pumps.

The continual flow of coolant to the reactor allowed the core to remain covered while residual heat was removed, a process that can take days to safely complete. However, a little more than an hour after the accident began, the coolant pumps started to vibrate continuously. Unaware that coolant was leaking out of the core, operators shutdown the coolant pumps in an effort to preserve the equipment. This resulted in the core becoming uncovered and zirconium fuel began to oxidize, producing hydrogen. The reactor core would melt and a small hydrogen detonation would occur. Eventually the coolant pumps were restarted and the core was able to cool. A small controlled release of radioactive gas was done in order to prevent further hydrogen reactions.

1.1.4.2 Response to Three Mile Island The length of the accident was relatively short. By the evening of March 28th, the coolant pumps had been restarted and the core was cooling properly. The small controlled release of radioactive gas mitigated the possibility of a hydrogen explosion threatening the integrity of the containment. However, the inability of the utility running the plant, as well as the local and federal government to properly and promptly explain the situation to the general public, led to severe actions. Media coverage of the accident was large and in some cases, there was a belief that a hydrogen explosion could pose a severe risk to the public, mostly due to the incorrect association with a hydrogen bomb. It is estimated that over 144,00 people were told to evacuate and surrounding schools were closed, days after coolant had been restored to the core.No injuries were attributed to the accident.

There has been a large debate on the root causes of the TMI accident. Early on, it was

argued that this event was a "Normal Accident" that was unpreventable due to the complexities of a nuclear power plant and the limitations of the technology. This argument has been challenged several times. One such study, published in the Journal of Contingencies and Crisis Management, found that management was a major contributor to this accident. This included poor communication of previous events, lack of training for the operators, etc [15]. This is beneficial for the nuclear industry, if the issues at TMI were unavoidable, it would be impossible for the public to have confidence in nuclear power. Since many of the issues were preventable, the industry has had the opportunity to improve. To its credit, the nuclear industry in the United States has accepted this accountability. Several reforms have been implemented by the utilities and the Nuclear Regulatory Commission (NRC), including expanding the use of human factors and simulations to improve training, better communication between plants to share lessons learned, approaching nuclear safety from a probabilistic approach and embracing the concept of continuous improvement when it comes to safety.

1.1.4.3 Use of Simulation in the Nuclear Industry One of the biggest changes in the nuclear industry after the TMI accident was the expanded use of simulators for nuclear power plants. The International Atomic Energy Agency (IAEA) has performed a number of studies on the use of simulation for different applications in the nuclear field. These areas include: fuels, accident planning, reactor behavior for new designs and training. The history of power plant simulation goes back to the to late 1950's, when simulators where used for steam systems to help with operator training. In 1968, General Electric (G.E) developed a complete power plant simulator for its Boiling Water Reactor (BWR) design[16]. In response to the TMI incident and the rapid growth in the use of computer technology, simulator use became widespread in the 1980's. In 1987, the NRC issued a requirement, 10 Code of Federal Regulations (CFR) Part 55.4, that all nuclear power plants operating in the United States must make use of a simulation facility onsite by May of

1991, for training purposes[17].

The IAEA has defined four different types of simulators for nuclear power plants: basic principle, full scope, other than full scope control room and part task[18]. Full scope simulators are designed to represent a specific reactor at a plant, including the human interface. These are usually used in simulation facilities at plants. Basic principle simulators are similar, but do not represent a specific reactor and do not necessarily use the same interface. Other than full scope simulators represent a specific reactor accurately, but use a different human interface than the actual reactor. For example, the SONGS simulator used at INL uses a touch screen interface, rather than the traditional controls of a reactor. Figure 2 shows the human interface from the SONGS reactor simulator at INL. Finally, a part task simulator only represents a specific component of a reactor, such as a piece of equipment.

Reactor simulators make use of several different computer software codes to accurately mimic actual reactor behavior. In an example given by INL, a simulator uses the RELAP code to model hydraulic behavior, a simulator platform working along with a plant specific model. This software is then integrated into the human interface. It should be noted that all code used to model a reactor's behavior must meet the requirements of the NRC if it is to be used in a nuclear application. This requirement falls under 10 CFR Part 50.55a, which outlines the Verify & Validate (V&V) process required for a code to be used in a simulation[19].

Today, simulators are fairly common tools used in the nuclear industry. INL used the SONGS simulator to help a facility better plan the layout for the plant's control room. INL's Human Factors Lab also makes use of simulators to study the operator behavior to improve training. This includes monitoring human performance, interaction with equipment and error recognition. Other countries which make significant use of nuclear power, also rely simulation. The Russian Federation has used Part Task simulators to study turbine behavior under different conditions[20]. France has made extensive use

Figure 2: SONGS Reactor Simulator[18]

of full scope in operator training and the evaluation of new training techniques and in the United Kingdom, simulators have been used to evaluate the management of control rooms.

1.1.5 Proposed Machine Learning Applications in Nuclear Safety

As is the case in many other fields, researchers have been exploring different applications for machine learning in nuclear safety. Due to the high number of regulations the major-

ity of these proposals are in beginning stages of development and have not been applied with operating reactors. This sections will look at many of the different case studies that have been done using machine learning in the area of nuclear safety.

1.1.5.1 Photo Recognition Over the last decade, there have been numerous efforts to use machine learning to help with preventative maintenance. One promising study took place at Purdue. This study proposed using photos of structural cracks to train a model, so it could identify cracks within a reactor. Machine learning models have history of being used for photo recognition applications. Using a model allows for more in-depth analysis that can catch things human eyes cannot[21]. Also, the use of photos allows for structural analysis to take place in areas that are difficult for humans to reach and analyze.This study used a Naive Bayes Convolutional Neural Network (CNN) based model and initial results have been positive. Initial accuracy in the classification was over 98% higher than that typically expected of humans. Figure 3 shows a sample of the photo used in the testing of the model for this study. Another study that has made use of the

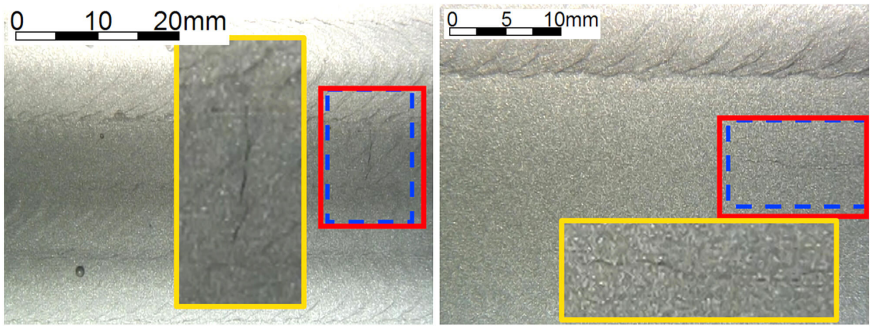


Figure 3: Sample Crack Photograph use in model testing[21]

photo recognition abilities of machine learning models for nuclear safety came from a collaboration between Oak Ridge National Lab (ORNL) and the University of Wisconsin-Madison. In this study, a neural network based model was trained using 270 electron microscopy images. From these images, the model was able to correctly identify images with material that had suffered damage from radiation exposure with an accuracy

of 86%. This was higher than the 80% expected accuracy from humans[22].

1.1.5.2 Fuels Machine learning has also been used in the area of fuel behavior. In a collaboration between ORNL and University of Illinois at Urbana-Champaign, a machine learning model was trained to determine the composition of fuel from a Pressurized Water Reactor (PWR)[23]. The data used to train this model was obtained through the use of a reactor simulator. The model trained was neural network based and reported a 95% accuracy. Fuel safety application of machine learning models have been proposed for a number of years. In 2003, a study done in Turkey at the Çekmece Nuclear Research and Training Center, proposed using a neural network to determine the optimal loading pattern for a PWR[24]. A better configuration of the loading pattern for a reactor can improve fuel efficiency and lead to savings in cost. In addition, the fuel assembly would be more reliable improving safety.

1.1.5.3 Human Factors & Decision Making Machine learning has had a large number of business related applications throughout the years. G.E, one of the world's leaders in reactor development, has been looking at different ways to apply this concepts to improve facility management and decision making[25]. The hope is that investment in this, will lead to safer and better run plants, which will result in improved cost efficiency. G.E is exploring the use of cognitive computing to give managers access to more useful information for decision making, which in turn, should improve plant efficiency. G.E is also looking at ways to improve data collection by using digital twins, which would not only speed up collection efforts, but allow for quicker diagnostics in plant equipment and better planning for maintenance. Other areas G.E is planning to implement machine learning include energy transmission and energy storage.

An effort between Massachusetts Institute of Technology and Tsinghua University looked at the use of regression models to aid in decision making. An Support Vector Regression (SVR) model was used for the project. The effort attempted to determine a

value to measure the performance of an advance reactor. This value would serve as an indicator to operators and managers. It is hoped this would aid decision makers in implementing adjustments to improve performance and efficiencies[26].

1.1.6 Transient Identification Case Studies

One of the most promising areas of research in the nuclear industry involving machine learning is the identification of transients events occurring with a reactor. As seen with TMI, the consequences of failing to properly diagnose a transient can be felt for years to come. There have been a number of studies in this area using a variety of different approaches.

1.1.6.1 Neural Network Based Projects In recent times, neural network models have become the most common type of model trained for diagnosing reactor transients. Some efforts have been looking at ways to use these neural network based models to develop automated responses to transient events. In a collaboration between Northeastern University and Nuclear Power Institute of China, an unsupervised learning model was developed to automatically using neural networks and pattern recognition[27]. Another similar project occurred at North West University in South Africa. Here, researchers worked toward using simulator data as reference when developing their model[28].

The United States Department of Energy (DOE) has shown great interest in developing more autonomous systems. In 2018, DOE awarded North Carolina State University (NCSU) a 3.5 million dollar grant to develop A.I. based systems for nuclear reactors[29]. One project to be funded by this grant involves identifying a Loss of Flow Accident (LOFA) transient, due to a failure of a sodium pump occurring with a sodium cooled fast reactor[30]. Data was collected using a simulator of the Experimental Breeder Reactor 2 (EBR-2) from INL. The system being designed will be able to take corrective action once it has detected that a LOFA has occurred. This is part of a Nearly Autonomous Manage-

ment System (NAMS) system. This project makes use of both, digital twins and neural networks, to train the model. Initial results have been positive and NCSU has begun developing self-learning algorithms to add to the NAMS.

Other efforts have made use of different types of neural networks to train a model. One of these took place at the University of Wisconsin-Madison. In this study, researchers trained a Recurrent Neural Network (RNN) model in an effort to try and address issues with uncertainties in the data, as well as time series issues with the sensors[31]. The hope was that by using RNN, the model could better deal with an unbalanced dataset. The unbalanced dataset was due to the lack of transient data and concerns over existing data. Results showed that the RNN model trained was better able to detect issues, even with an unbalanced dataset. paragraph CNN's have also been proposed as a neural network technique for diagnosing reactor transients. A study from the Chinese Academy of Sciences used a combination of a CNN and small batch processing to make the diagnosis[32]. The accuracy of the trained model was encouraging with an average result around 90%. Another type of neural network that has been looked at is a Back Propagation Network (BPN). One of the first studies involving BPNs goes back to 1995. Researchers at Japan's Hokkaido University, proposed using BPN to diagnose two transients within the Jōyō experimental reactor[33]. A more recent study of the use of a BPN occurred at Korea Institute of Science and Technology. This study found that a BPN could diagnose a single transient with few issues[34].

The concept of deep learning models has also gain a lot of interest in reactor transient diagnostics. At Federal University of Rio de Janeiro in Brazil, researchers collected data from a PWR simulator to train the deep learning model. The model was also given the option of "don't know" in making a classification. Results were positive with validation measurements near 95%[35].Another study that made use of a deep learning model for reactor safety occurred at the Korea Atomic Energy Research Institute. The goal of this study was to develop a model that could detect the transient in the early stages, before

an alarm signal is triggered[36].

1.1.6.2 Support Vector Machine Models Although the majority of research in transient diagnostics with machine learning has focused around neural networks, some studies have looked at the potential of using SVM models to identify a transient. The University of New Mexico, Tong University and Seoul National University develop a SVM model to diagnose a LOFA transient. Data was collected by running several different LOFA simulations using the Multi-dimensional Analysis of Reactor Safety (MARS) code. Results were positive with the model able to determine if a LOFA is occurring, assuming enough training data is provided[37].Another study that made use of SVM came from the Royal Institute of Technology in Sweden. In this project, researchers created four separate SVM models for four different transients. The results were positive, as each of the four models could predict the transient nearly perfectly[38].

1.2 Project Overview

1.2.1 Project Goals

The primary goal of this part of the project is to develop a machine model that can predict transients occurring within a nuclear reactor. The development of such a model can contribute to the development of data driven safety systems for nuclear reactors. Data driven systems should lead to quicker, more effective responses to transient events, which will reduce downtime for nuclear reactors. This in turn, would create the potential for great economic benefits. Consider a 5000 megawatt thermal/ 1100 megawatt electric nuclear reactor. This size of plant can produce over 26,000 megawatt hours a day, if the plant is run at max capacity. In May of 2019, the average cost of electricity in the United States was 10.42 cents per kilowatt hour or 104.20 dollars per megawatt hour[39]. If a reactor was to experience a transient event and be shutdown by the NRC, the plant would be losing over 2.5 million dollars per day in revenue. An unplanned shutdown could result

in months of shutdown and millions in lost revenue. Quicker response to the transient, or prevention, could help get the plant back into operation faster through quicker repair times and responses to regulator inquires.

Many machine learning studies in the area of reactor transient diagnostics envision the models trained to be part of an automated system that will control responses to events within the system. For example, the project at NCSU is centered around autonomous responses to a LOFA event. Although there are many benefits to automation, there are many issues surrounding what degree of control an automated system should have versus the people managing the system. This has been an ongoing issue in the many industries such as aerospace where issues in automation and lack of training and understanding of the automation by those operating the system has led to accidents. It is important to note that this is not the goal of this study.

The focus of this study is aiding in the development of a system that can assist reactor operators in performing their duties rather than altering the role of the operator. It is hoped that this study can contribute to the development of a data based system that can provide operators with a quick and accurate diagnosis on possible transient events occurring within a reactor system. The operators could then make use of this information to more quickly verify the initial diagnosis and take corrective action. This could also allow for stakeholders, including the general public, in the facility to be more quickly informed of issues at the plant. If the flow of information from the plant improves flow, it is possible there will be an increase in public confidence in nuclear generated electricity.

The development of a system to aid reactor operators would still have to be approved by regulatory agencies, such as the NRC. Still, as the system will have no direct control of any nuclear related systems, the approval and evaluation process would likely take significantly less time than a system that has some degree of control on the system. Lessons learned through this development process may also aid in the development of more advance automotive systems not only in the nuclear industry, but in other areas where au-

tomation is a topic of interest.

1.2.2 Objectives

The following list summarizes the major objectives of the work performed in this experiment.

List of Project Objectives

1. Train a number of machine learning models to correctly classify five different events within a reactor system.
2. Expand the initial dataset and re-train models to correctly classify 12 different events.
3. Perform analysis on whether the optimal machine learning models can lose significant features and still make a correct classification.
4. Study the impact changes in the random state used to train the optimal machine learning models has a significant impact on model performance.
5. Explore the use of anomaly detection techniques to see if a machine learning model can identify significant anomalies within a dataset.
6. Explore the performance of different AutoML Packages including model performance, ease of use, etc.

1.2.3 Novelty

Over the last decade, several studies have been done on the use of machine learning models to diagnose a transient event occurring with a nuclear reactor. The vast majority of these studies have used some form of neural network based model. There is merit to this, as neural networks are generally highly accurate. Also, neural networks have the ability to handle large datasets, especially those with a high number of features. A number of the studies reviewed for this project used over one hundred features in the models trained. There are some drawbacks however to neural networks. First, neural networks can become complex quickly, as the number of layers and neurons is increased.

This complexity makes neural network models difficult to visualize or explain mathematically. This has led to a perception that neural networks act as a "black box". Due to this, it is likely that any neural network based safety system will have to undergo a long regulatory review before it can be used in an actual reactor. One of the key differences in this project and the studies reviewed is the use of traditional machine learning models, such as decision trees and logistic regression. Until the first phase of this project, no study had been done using these types of models for transient diagnosis. Models like decision trees, can be more easily visualized, which would help in a regulatory review. Other models, such as logistic regression or naive Bayes, are probabilistic in nature and can be easier to explain mathematically.

The next key novelty for this project is the use of AutoML tools to train the model. AutoML is a relatively new approach. Most common AutoML packages used today were developed in the last decade. As a result, no studies in the nuclear field have made use of an AutoML package, such as TPOT. The use of AutoML not only can simplify the process, but allow a more comprehensive search for the optimal classification model. Combining a tool like TPOT with the HPC capabilities at INL can create a fast, effective and less complex model for diagnosing transient events. It should also be noted, that few studies have been done comparing the effectiveness of different AutoML packages. This may be an ideal opportunity to do such a study.

An area that also has not received much attention in studies that attempt to create a model to diagnose transient events is model redundancy. This could be a vital part of any type of machine learning based safety system. Machine leaning models rely on data for both, training and testing. Without data, the model will not be effective. Transient behavior can be simulated, however, the impact on equipment such as sensor is more unpredictable. It is possible that one or more sensors are damaged during a transient. This would make models that rely on the data provided by the sensor ineffective. Another possibility is that an outside attack could alter data from the sensors. This could also

result in the models becoming ineffective. For this reason, it is essential that data be scanned for anomalies prior to use it with a machine learning model in practice. Also, to deal with the potential of some data being unavailable or unreliable, it is essential that additional models which rely on fewer features be trained along side the optimal model. The proposed decision tree analysis will help meet these goals.

1.2.4 Summary of Novelty

Points of Novelty for Proposed Study -

1. Use of traditional machine learning models to identify transients.
2. Use of AutoML package TPOT to identify an optimal model.
3. Analysis of the decision tree and random forest models from TPOT to perform a study on model redundancy.
4. If possible, perform a study comparing different AutoML packages.

2 Tools and Methods

Due to the scope of this project, a number of different tools will have to be used for both data collection and for training the machine learning models. In addition, this project makes use of several different approaches to train models and process data. This section will go over the tools used to collect data, the software used to perform the project and the techniques utilized to train the models.

2.1 Python Packages

Today there are a number of different programming languages that have machine learning packages. This includes Shark in C++, DL4J in Java and DataExplorer in R. Perhaps, the most widely used language for machine learning is Python. This project will make extensive use of several different Python packages. In this section an overview of the history and capabilities of the packages used for this part of the project will be given.

2.1.1 NumPy

NumPy is a Python library that has been designed with several different applications. These include: the ability to integrate C/C++ and FORTRAN code into Python, preform linear algebra functions and easy integration with datasets. NumPy was developed by Dr. Travis Oliphant in 2005, as a successor to Numeric and Numarray. NumPy development is done through the NumFocus Foundation, a nonprofit organization. Today, NumPy is used by many companies and organizations, such as Netflix and NASA [40]. NumPy was designed with scientific computing in mind, but the package also has the ability to help with database construction and manipulation. NumPy is a free open-source package and is included standard in many Python distributions, such as Cygwin and Anaconda. It is also standard on most IDE libraries such as PyCharm. The library can be downloaded using Pip. Many of the packages used in this project, such as TPOT and

Pandas, use NumPy arrays to perform numerical operations. At the time of this part of the project, the most current version of NumPy was version 1.17.0, which was released in May of 2019. NumPy is able to operate in a Windows, Linux or Mac OS environments[41].

One of the most important features in the NumPy library is the ability to create NumPy arrays, also referred to as ndarrays. A NumPy array is a container that allows for the storage of several different elements. NumPy arrays, while similar to a Python list, have a few key differences. The first is the ability to operate quicker and take up less memory than a Python list. This is due to better integration with C/C++, which helps mitigate the loss of efficiency that higher-level and easier-to-use languages typically have. Also, NumPy has been optimized for linear algebra operations. NumPy arrays are considered homogenous, meaning that all data is the same size and is processed the same way, regardless of any differences between elements. These elements are described by a dtype object that can be built using different data types. Every NumPy array has a dtype object associated with it. This can tell the user the descriptive information about the NumPy array, such as type, memory usage, etc. Elements used in code are taken from the array using indexing. The index represents an object scaler which was part of the NumPy development.

2.1.2 Pandas

Pandas is a free, open-source Python package, which aims to help users with data manipulation, modification and analysis. The package can be downloaded on most Python distributions, such as Anaconda or via Pip. It can also be added via most Python IDEs. The package was initially developed by Wes McKinney in 2008 in response to a need for better data tools. Development is ongoing as of 2021. The project receives funding and support from the University of Paris Saclay Center for Data Science, as well as from Two Sigma. Pandas required Python version 2.7 or greater but support for all Python 2 versions was dropped on January 1st of 2020, as such Python 3 or higher is required to use Pandas. The most current version at the time of this phase of the project was version

0.24.1, released in February of 2019. This was the version used in this part of the project. Pandas only requires the NumPy package to operate properly. Pandas is designed to work with Windows, Mac OS and Linux environments[42].

The goal of the Pandas project is to provide data tools to users in Python. In the past, the adaptation of Python in data science and statistics had been slow as users had preferred to use tools such as MATLAB and R. Pandas has the ability to read and convert datasets, typically in a CSV format, into a structured dataset known as a Pandas DataFrame. A Pandas DataFrame is a 2-dimensional Python list that allows users to store values in a tabular form. An example of a DataFrame is shown in Figure 4. Like most Python packages, Pandas is considered high-level and there are tradeoffs in efficiency for ease-of-use. Cython was used to mitigate this issue. Pandas has the ability to help the user identify and manage missing values, a common issue in data science. The package also has the ability to group or sort data by specified user input, such as individual values within the dataset or data types like floats or strings. Other useful Pandas functions include the ability to change large groups of data, perform statistical analysis such as mean, median and standard deviations over the dataset, as well as add, remove and combine parts of different datasets [43]. Figure 4 shows a part of the DataFrame used in this project.

	0	1	2	3	4	5	...	18	19	20	21	22	23
0	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.55	652.741	1406.51	EOL	Transient-Feedwater-Pump-Trip
1	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.55	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
2	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
3	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
4	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.742	1406.52	EOL	Transient-Feedwater-Pump-Trip
5	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.742	1406.52	EOL	Transient-Feedwater-Pump-Trip
6	617.594	559.384	559.384	617.627	559.388	559.388	...	588.498	2235.57	652.742	1406.53	EOL	Transient-Feedwater-Pump-Trip
7	617.594	559.384	559.384	617.627	559.388	559.388	...	588.498	2235.57	652.742	1406.53	EOL	Transient-Feedwater-Pump-Trip
8	617.594	559.384	559.384	617.627	559.389	559.389	...	588.498	2235.57	652.742	1406.54	EOL	Transient-Feedwater-Pump-Trip
9	617.594	559.384	559.384	617.627	559.389	559.389	...	588.498	2235.57	652.742	1406.54	EOL	Transient-Feedwater-Pump-Trip
10	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.742	1406.55	EOL	Transient-Feedwater-Pump-Trip
11	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.743	1406.55	EOL	Transient-Feedwater-Pump-Trip
12	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.743	1406.56	EOL	Transient-Feedwater-Pump-Trip
13	617.594	559.385	559.385	617.627	559.389	559.389	...	588.499	2235.59	652.743	1406.57	EOL	Transient-Feedwater-Pump-Trip
14	617.594	559.385	559.385	617.627	559.389	559.389	...	588.499	2235.59	652.743	1406.57	EOL	Transient-Feedwater-Pump-Trip
15	617.594	559.385	559.385	617.627	559.389	559.390	...	588.499	2235.59	652.743	1406.58	EOL	Transient-Feedwater-Pump-Trip
16	617.595	559.385	559.385	617.628	559.390	559.390	...	588.499	2235.59	652.744	1406.59	EOL	Transient-Feedwater-Pump-Trip

Figure 4: Sample DataFrame from Project

2.1.3 Scikit-learn

This project makes great use of the scikit-learn Python package. The package is a free, open-source package that can be downloaded through Pip or a Python distribution, such as Anaconda. Scikit-learn was developed by Dr. David Cournapeau in 2007, as a summer project for a Google Summer of Code Project. The purpose behind the project was to design a system that could run complicated machine learning algorithms using Python and maintain a user-friendly intuitive interface. The first public release of scikit-learn was released on February 1st, 2010 and the French Institute for Research in Computer Science and Automation (INRIA) began heading the project. Today, scikit-learn development and research is funded by universities, such as the New York University, University of Sydney, and Columbia University, among others [44]. Many companies use scikit-learn as part of their information system operations, including: JP Morgan, Spotify, Booking.com and Change.org. Some of these applications include predicting user's preference in music, credit and market trend analysis, as well as targeting users with more customized add-ons and specials [45]. Similar to many other Python packages, scikit-learn makes use of many modern C++ libraries using Cython, a programming language designed to help bridge C and Python code. Scikit-learn has been designed to be compatible in both, a Windows or Linux environment. The latest version of scikit-learn 0.21, the version used at the time of this part of the project, requires Python version 3.5 or higher. Scikit-learn relies on three Python packages to run, NumPy, SciPy and Joblib, which allows the package to be easily distributed and used. While not required, Pandas is needed in order to take full advantage of the abilities of the scikit-learn package. In the past, the package has focused on remaining easy to use and efficient, rather than adding new features. Though recently, scikit-learn has been updated with new features that assist in data exploration using Pandas. This includes better ways of dealing with missing values with the SimpleImputer function. Since Python is a high-level programming language, there are tradeoffs in code efficiency for ease of use. Steps have been taken in

order to manage and mitigate most of these issues: the specification of objects through interface rather than inheritance, the use of Cython to increase the efficiency of using C++ libraries within Python, and others [46].

Scikit-learn has the ability to perform several different types of machine learning algorithms: supervised learning methods, such as classification and regression, as well as unsupervised methods, like k-means clustering. Currently, scikit-learn has functions to perform more than 17 different types of supervised machine learning methods, as well as 9 different unsupervised methods. The package has been designed with functions that help with data preparation, such as splitting datasets for validation purposes. Scikit-learn also has several functions for data preprocessing, such as the standard scaler function and model validation and scoring through measurements, such as accuracy, precision, and goodness of fit. Finally, scikit-learn can make use of Python's Matplotlib package in order to help users visualize the results of the models generated. This includes clustering graphs, confusion matrices, etc[47].

2.2 AutoML

The process of training machine learning models can be complex. Even with the large number of tools available, such as scikit-learn, training models can be a time consuming process. This is because even for training a single type of model there are a number of pre-processing techniques, feature selectors, etc. that can be applied to try and improve the model. The only way to find the optimal model is to train multiple models using the different combinations of pre-processing techniques. Keeping track of all these results, in addition to the different combinations, can add a significant amount of time to any project. However, recently a new approach for machine learning known as Auto machine learning or AutoML has looked to address this issue. While this simplifies the process, it can be computationally expensive. Python packages that are currently available include: Auto-WEKA/auto-sklearn, H2O and TPOT. Google has also developed its own

cloud-based AutoML software, called Cloud AutoML, to try and open machine learning to non-data-scientists. Services available include photo and video analysis/modeling, language translation and data analysis [48].

2.2.1 TPOT

The supervised learning models for this project were created using the Tree-based Pipeline Optimization Tool (TPOT) package in Python. TPOT was chosen for this project because it is one of the more mature AutoML Python packages available. Also, it is simple to use and can evaluate a number of different machine learning models. TPOT was developed by the Computational Genetics Lab at the University of Pennsylvania with support from the National Institute of Health. Development began in 2011 and the package continues to be developed by Epistasis Lab at the University of Pennsylvania. TPOT is open-source and is available for download from the lab's GitHub repository for free. TPOT was developed in response to the growing demand and interest in machine learning applications. The process of creating a machine learning model can be complex and time consuming, even if just limited to supervised learning. There are multiple models that can be created, as well as many different methods of data preprocessing. It can be difficult for even an experienced data scientist to develop the best possible model. The purpose of the TPOT package is to simplify and automate parts of the machine learning process, while providing better results due to improved data preprocessing and the use of multiple different supervised learning methods[49]. TPOT is designed to make use of the scikit-learn Python package for both, data preprocessing and model construction. As such, the user is required to have the scikit-learn package installed and imported into the program. TPOT also makes use of NumPy arrays and Pandas DataFrames and these packages are required as well. The DEAP, SciPy, tqdm, stopit, and update_checker packages are also needed. These packages are all available for free via download and can be configured with the Anaconda Python distribution using Pip. Other Python distributions

can be used if the pywin32 module is used. It should be noted that Epistasis Lab strongly recommends that Python 3 be used rather than Python 2. TPOT is designed to aid the user in data preparation for supervised learning. This includes automating feature pre-processing, selection and construction. Doing so can dramatically simplify the process of preparing data for use in machine learning algorithms and these steps can be situational, complex and time consuming. It is important to note that the TPOT package requires the user to do data examination on the data to be used in the supervised learning process. The package cannot account for missing values, qualitative data and incorrectly formatted datasets.

Intially, TPOT only had functionality with machine learning models and techniques from the scikit-learn library. However, in recent years support has been added for a number of other libraries. These have been added to allow TPOT the ability to train models using neural networks. To do this, TPOT has an optional dictionary, TPOT NN, that can be enabled that makes use of a number of Facebook's PyTorch neural network models. TPOT has also added functionality to better make use of GPUs. In 2020, a new dictionary, TPOT cuML, was added that allowed for TPOT to make use of some of Nvidia 's CUDA techniques. It should be noted, that at this time the package is very new and only has functionality with certain Nvidia hardware and only can make use of a limited number of machine learning models. Future updates are planned. This study will only make use of the scikit-learn library.

One of the advantages of TPOT is that the user can define a dictionary to specify which types of models will be training, as well as which data pre-processing techniques will be used in the training. Currently, this is only supported for models and functions from the scikit-learn library. Once the pre-processing has been completed, TPOT can train and test several different models using the reprocessed data. The process of training a model with the defined data pre-processing techniques is refereed to as a pipeline[50]. Figure 5 shows a visualization of the pipeline creation process. TPOT has the ability to train both,

regression and classification models.

nbNewBrun,

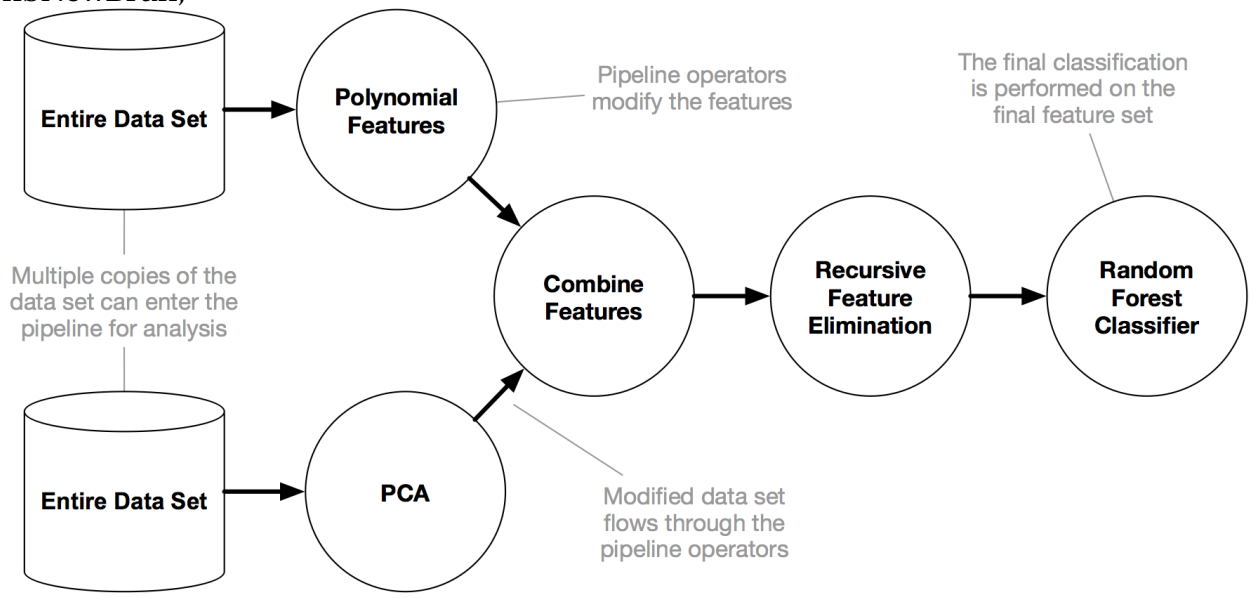


Figure 5: Typical TPOT Pipeline[50]

TPOT allows the user to define a number of parameters for model training. The first of these is the number of generations that will be used in the model creation. This is the number of iterations that will be used in the optimization process. The more generations run, the better the model results will likely be. However, this will also increase computation time. The next parameter that can be defined is the population size used. This is the number of individual pipelines retained in each generation. As was the case with the number of generations, the more pipelines retained will produce better results, but computation time will also increase. Users can also specify the random number seed and the verbosity used in training the model. TPOT also supports cross validation to help address the randomness of split data. Unless otherwise stated, it should be assumed that all models trained using TPOT for this project used 100 generations with a population size of 100. The cross validation will be set to 10 and the random state will be set to 0. A sample TPOT configuration is shown in Figure 6.

Once the TPOT classifier has been configured, the program will begin testing the different model combinations. This process can be very slow, especially if the dataset is

```
#Entering the TPOTClassifier Parameters
tpot = TPOTClassifier(generations=1, population_size=1, cv=10, verbosity=2,
random_state=5, config_dict=tpot_config)
```

Figure 6: Sample TPOT Classifier

large and/or a large number of parameters has been specified. TPOT will by default provide the accuracy of the optimal model, but scikit-learn methods can be used to produce the other validation measurements, if needed. The user can save the optimal pipeline for future use once the training has been completed. The optimal results can be stored for further use in the program and optimal pipeline can be exported for use at a later time. It should be noted that the user will need to ensure that the correct file name is used to import the data in the exported pipeline, as the exported file will only use a placeholder for this. Also, depending on data configuration, some slight modifications to the optimal model code may be needed to ensure that the data is properly read by the exported file. It should be noted that while TPOT can simplify the data pre-processsing, training and hyper parameter tuning of machine learning models, the user is still responsible for the data exploration and wrangling of the dataset. For example, the user must still identify any outliers or missing values in the set and address those prior to using TPOT. Figure 7 from the TPOT documentation shows the function of TPOT in the model training process.

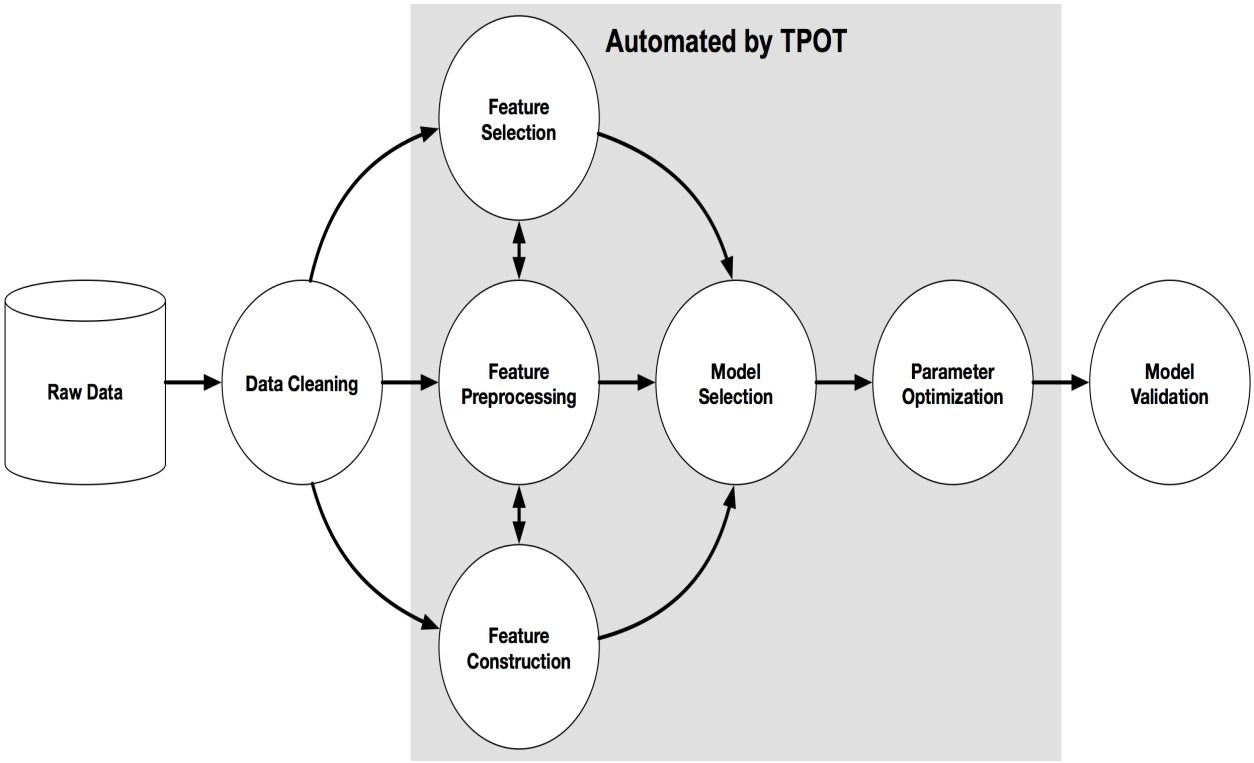


Figure 7: Training Process using TPOT [49]

2.3 Data Preprocessing Methods Used

In machine learning data preprocessing is an essential step in training models. Data scaling, feature selection and feature reduction are all important considerations when preparing a dataset for training. If data is not properly handled before beginning training, it is likely that the quality of the model will suffer. For example, in the dataset collected for this project there are various features including different temperatures, pressures, etc. These can be orders of magnitude different and using features that are not scaled may result in biased models. Proper data preprocessing can help manage this issue, however, there are many different techniques that can be employed. Also, there are no set rules on which techniques work better than others. This is completely dependent on the dataset. The only way to know which methods work best is to test and compare all of the different combinations. To better manage this, six different data scalers, three

feature selectors and three feature reduction techniques were defined in a custom TPOT dictionary. These were: Binarization, Feature Agglomeration, Maximal Absolute Scaling, Minimum-Maximum Scaling, Normalization, Principal Component Analysis, Robust Scaling, Standard Scaling and RBF kernel sampling. These are summarized in Table 1. This chapter will go into detail on the different data preproecessing techniques that were used in this project.

Table 1: Data Prepossessing Techniques used in TPOT Pipeline Creation

Binarization	Maximum Absolute Scaling	Robust Scaling
Standard Scaling	Minimum Maximum Scaling	Normalization
Principal Component Analysis	Feature Agglomeration	RBF Kernel Sampling
Variance Threshold Selection	Select Family Wise Error	Select Percentile

2.3.1 Binarization

Binarization is the process of taking numerical data and converting it into binary/boolean form. In machine learning, this method is most commonly used in preparing data for use with the Bernoulli naïve Bayes method, as this method requires data to be distributed in a binary form. There are other cases where this method could be applied. Binarization of data is particularly important in training image analyzers, where pixels are assigned a true or false value, based on the characteristics of the pixel[51].

Binarization relies on determining a threshold for the data that is to be converted. This threshold is the standard that determines if the data being converted is classified as a 1, representing a true value, or a 0, representing false value. The threshold value is dependent on the data and context of the analysis being performed. The scikit-learn has a preprocessing method .Binarizer(). This method will convert the user-specified data into binary form. By default, the threshold value is set to 0. As such, a negative or 0 value will be assigned a 0 and a positive value will be assigned a 1. The user is able to change this threshold so that the method better fits the analysis. Figure 8, shows a simple example

of the binarization method from the scikit-learn documentation[52].

```
Original Data
[[-1, -2, -3, -4, -5], [-4, -2, 0, 2, 4], [2, 4, 6, 8, 10]]
Transformed Data
[[0 0 0 0 0]
 [0 0 0 1 1]
 [1 1 1 1 1]]

(base) c:\ML\Figures>
```

Figure 8: Data Transformation Using Scikit-Learn's Binarization Function

2.3.2 Standard Scaling

One of the most common data scaling techniques used in preparing data for machine learning training is Standard Scaling.Standard Scaling removes the mean and then scales the dataset to its unit variance. The scaling of the individual data points, z, is given by the following equation:

$$z = \frac{(X - \mu)}{S}$$

Where: μ is the average of the dataset and S is the standard deviation of the dataset. Standard Scaling is necessary for many machine learning algorithms that require centered data. Also, this prevents bias from features that are of different type. The scikit-learn Standard Scaling function .StandardScaler(). Figure 9 and 10 show a comparison of unscaled and standard scaled data.

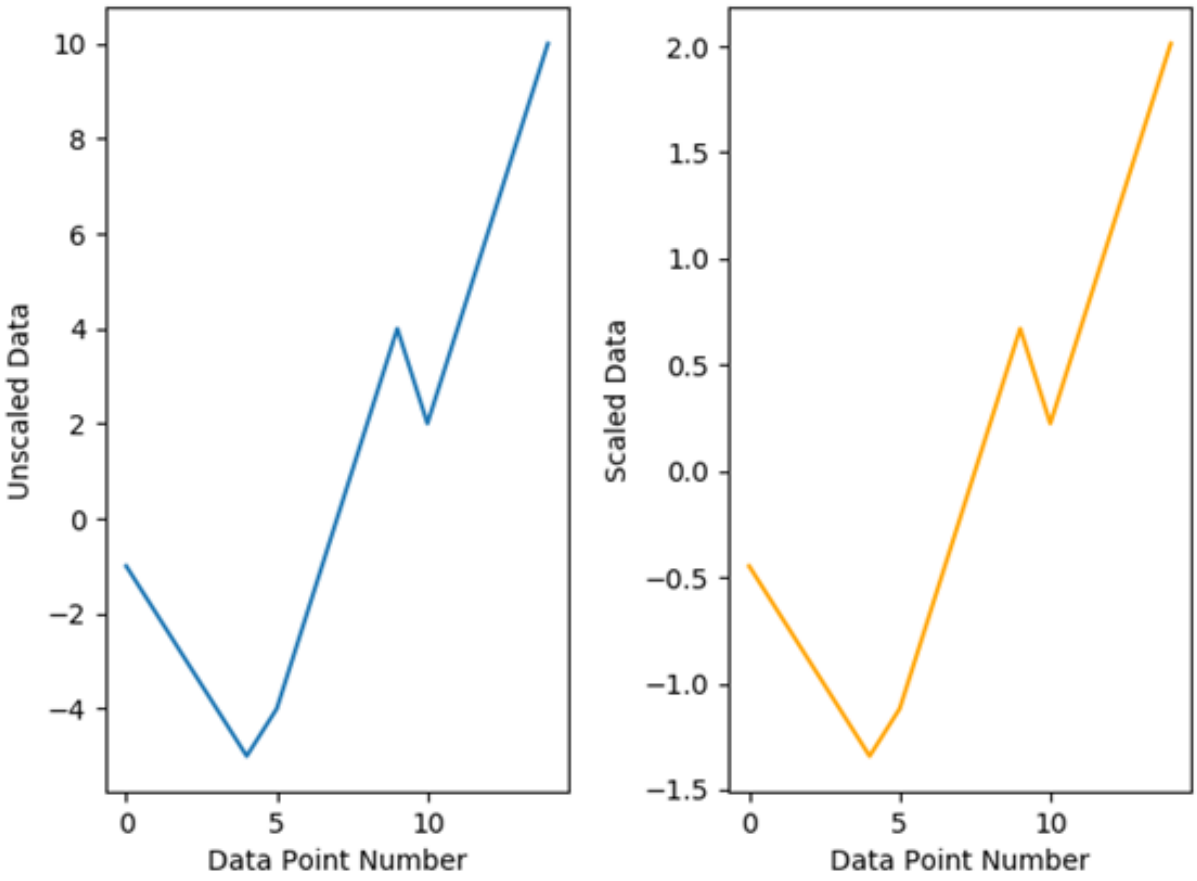


Figure 9: Comparison of Standard Scaled Data and Unscaled Data

```
Unscaled Data                Scaled Data
[[-1]                         [[-0.4472136 ]
 [-2]                         [-0.67082039]
 [-3]                         [-0.89442719]
 [-4]                         [-1.11803399]
 [-5]                         [-1.34164079]
 [-4]                         [-1.11803399]
 [-2]                         [-0.67082039]
 [ 0]                         [-0.2236068 ]
 [ 2]                         [ 0.2236068 ]
 [ 4]                         [ 0.67082039]
 [ 2]                         [ 0.2236068 ]
 [ 4]                         [ 0.67082039]
 [ 6]                         [ 1.11803399]
 [ 8]                         [ 1.56524758]
 [10]]                        [ 2.01246118]]
```

Figure 10: Numerical Example of Scaled Data

2.3.3 Robust Scaling

Another data scaling method used in the TPOT dictionary was the Robust Scaler. The Robust Scaler method scales features using statistical methods. These methods are the unit variance or the standard deviation of the feature being scaled. These methods are intended to help deal with outlying data points to prevent machine learning models from becoming skewed. This is done by removing and storing the median of the feature, and then the scaling range is calculated using 1st and 3rd quartile ranges as bounds by default. The user is able to adjust these ranges to better meet the needs of the dataset. The final value of the scaled data point is calculated using the same equation as the Minimum Maximum Scaler, just with the chosen statistic, rather than the actual value. Figure 11 shows the output of the scaling code. Figure 12, shows a comparison of data that is scaled using the Robust Scaling method and data that is unscaled.

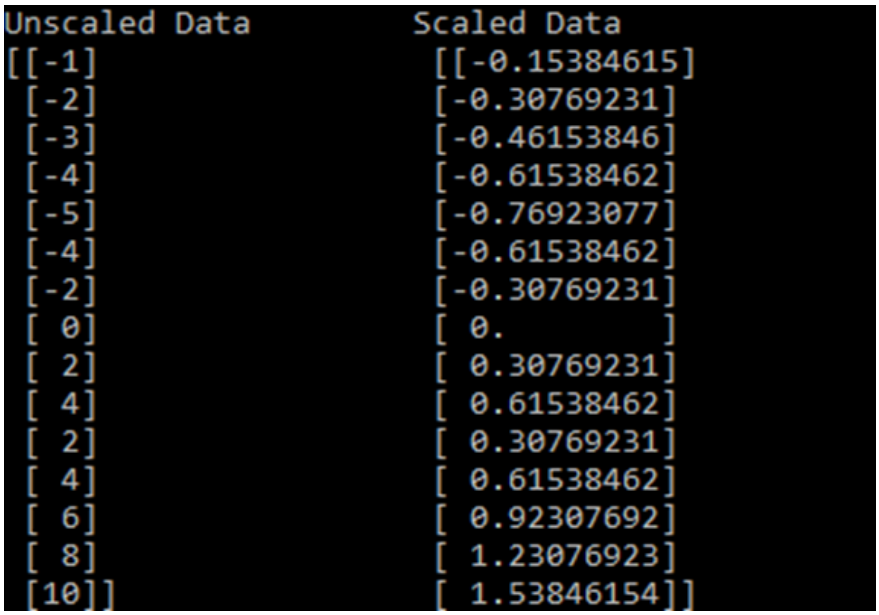


Figure 11: Numerical Example of Robust Scaled Data

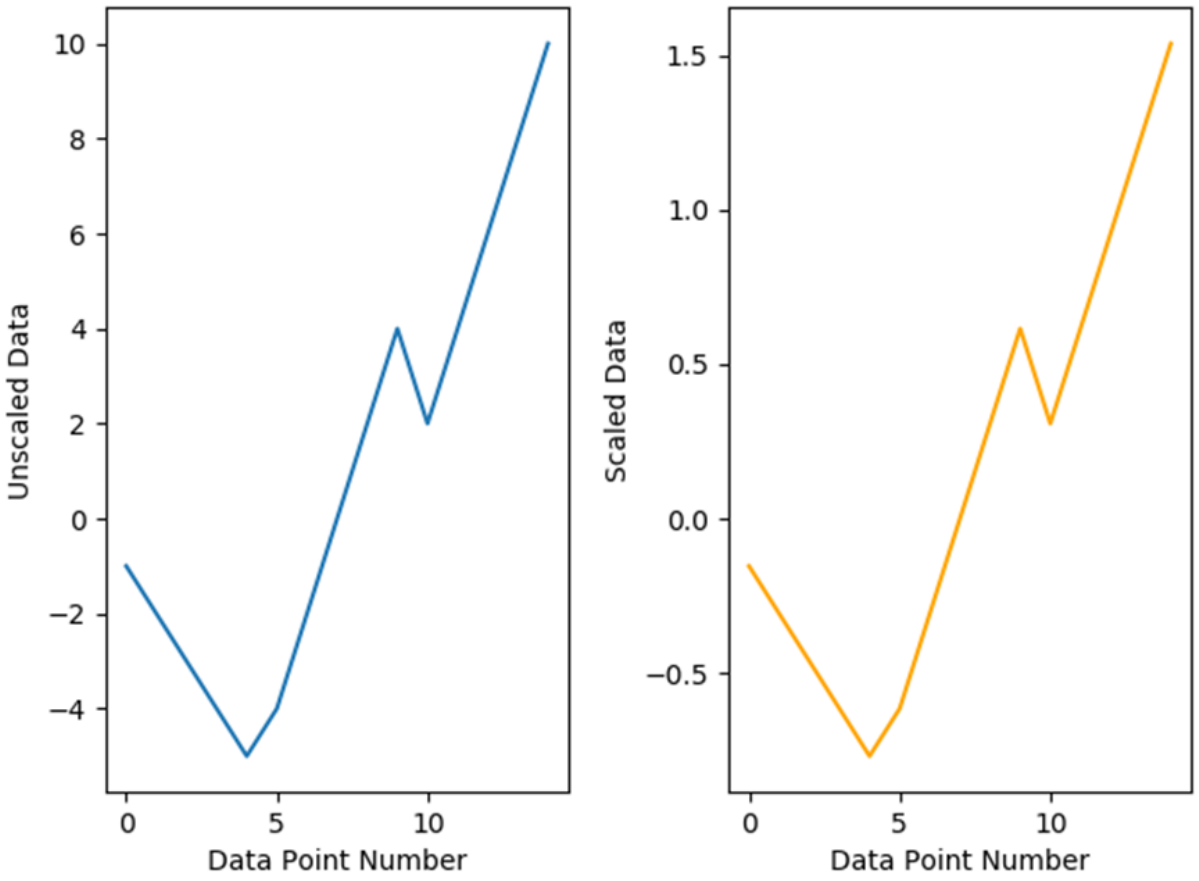


Figure 12: Comparison of Robust Scaled Data and Unscaled Data

The scikit-learn function for robust scaling is .RobustScaler(). The user can adjust the quartile range for the scaler and control if the data is centered prior to scaling. By default and for this project, the .RobustScaler() method will use the unit variance. The following equation is used for the .RobustScaler() method:

$$Scaled x_i = \frac{(x_i - 1stquatrtilerange(x))}{(3rdquartilerange(x) - 1stquartilerange(x)}$$

2.3.4 Maximum Absolute Value Scaling

The Maximum Absolute Value Scaler method will scale and translate the values in a dataset so the max value in that set is 1. Scaling in this way helps manage bias that results if the features are of different measurements, such as temperature and pressure. One issue that needs to be considered is that this scaling method is sensitive to outliers and failure to properly deal with this may result in skewed machine learning models. The scikit-learn method, .MaxAbsScaler() will ignore missing values and they are not changed during this process[53]. Figure 13 and 14 show a comparison of data that has not been scaled and data that has been scaled using the Maximum Absolute Value Scaler method.

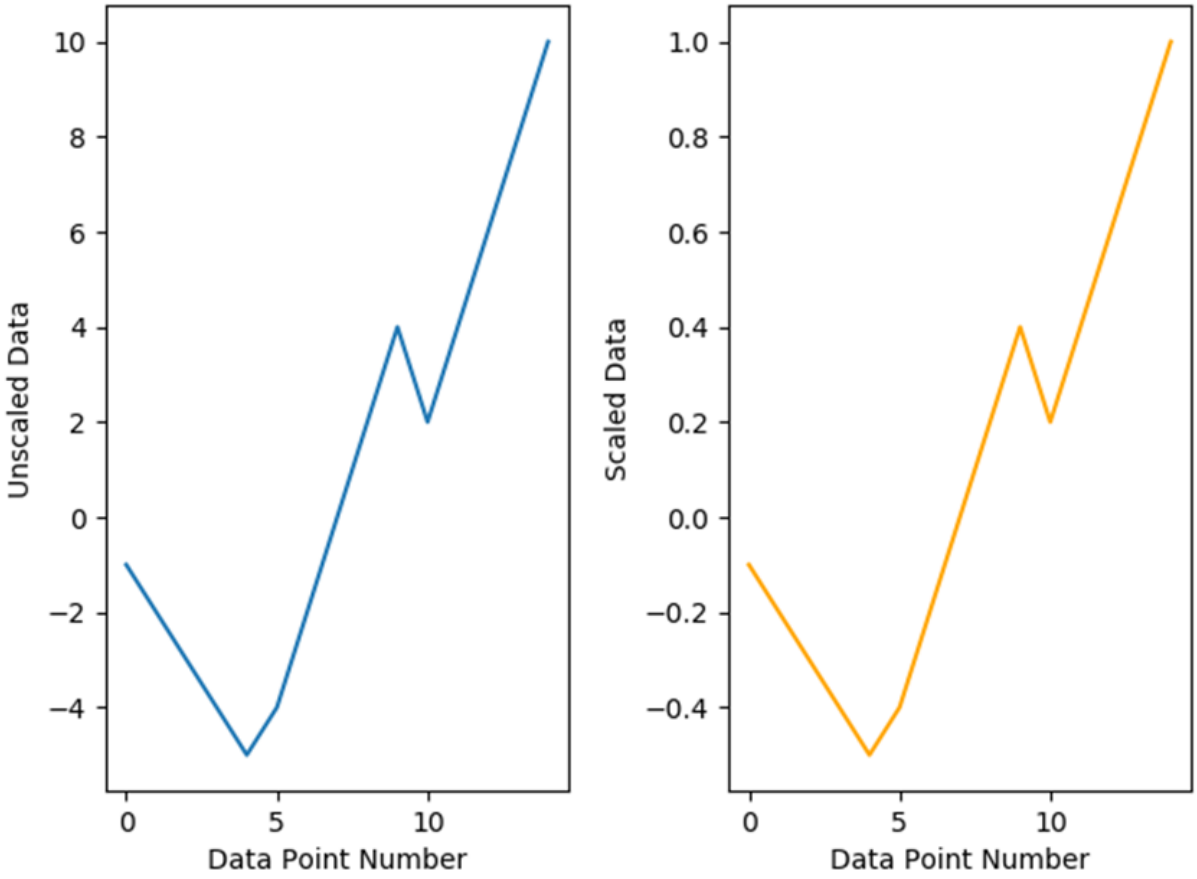


Figure 13: Comparison of Maximum Absolute Value Scaled Data & Unscaled Data

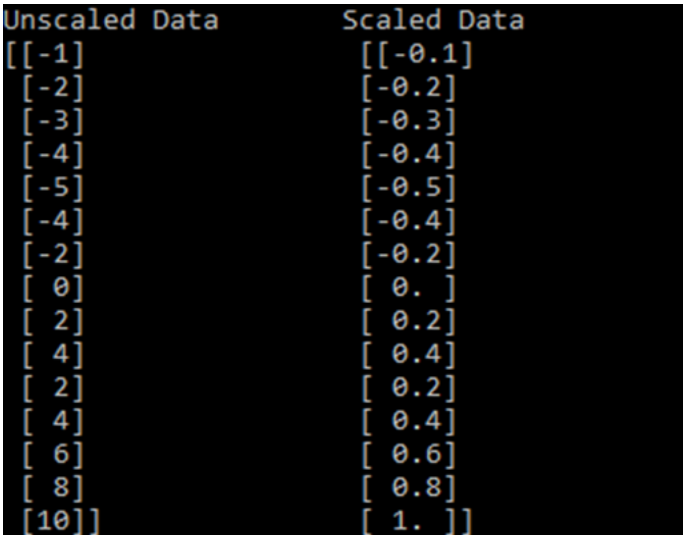


Figure 14: Numerical Example of Scaled Data Using Max Absolute Scaling

2.3.5 Minimum Maximum Scaler

The Minimum Maximum Scaling method allows the user to scale the data to a particular range. This range can be specified by the user to better fit the needs of the analysis. This project used the default range of 0 to 1. Each feature is scaled individually using the following equations:

$$X(Scaled) = X(Standard) * (MaximumBound - MinimumBound) + MinimumBound$$

Where:

$$X(Standard) = \frac{(X - X(Minimum))}{(X(Maximum) - X(Minimum)}$$

This method, like Maximum Absolute Scaler method, is useful to scale features that are of different measurements to reduce feature bias in the model. This method is also sensitive to the presence of outliers which could lead to skewed machine learning models if not dealt with properly. Also, this method is useful for the multinomial naïve Bayes algorithm, as it scales negative values to values between 0 and 1, which are compatible

with the algorithm. The scikit-learn method for the Minimum Maximum Scaler is .MinMaxScaler(). This method will ignore and leave untreated missing values from a dataset. Figure 15 shows a comparison of data that has been scaled using the .MinMaxScaler() method and unscaled data. Figure 16 shows the code output using the .MinMaxScaler() method.

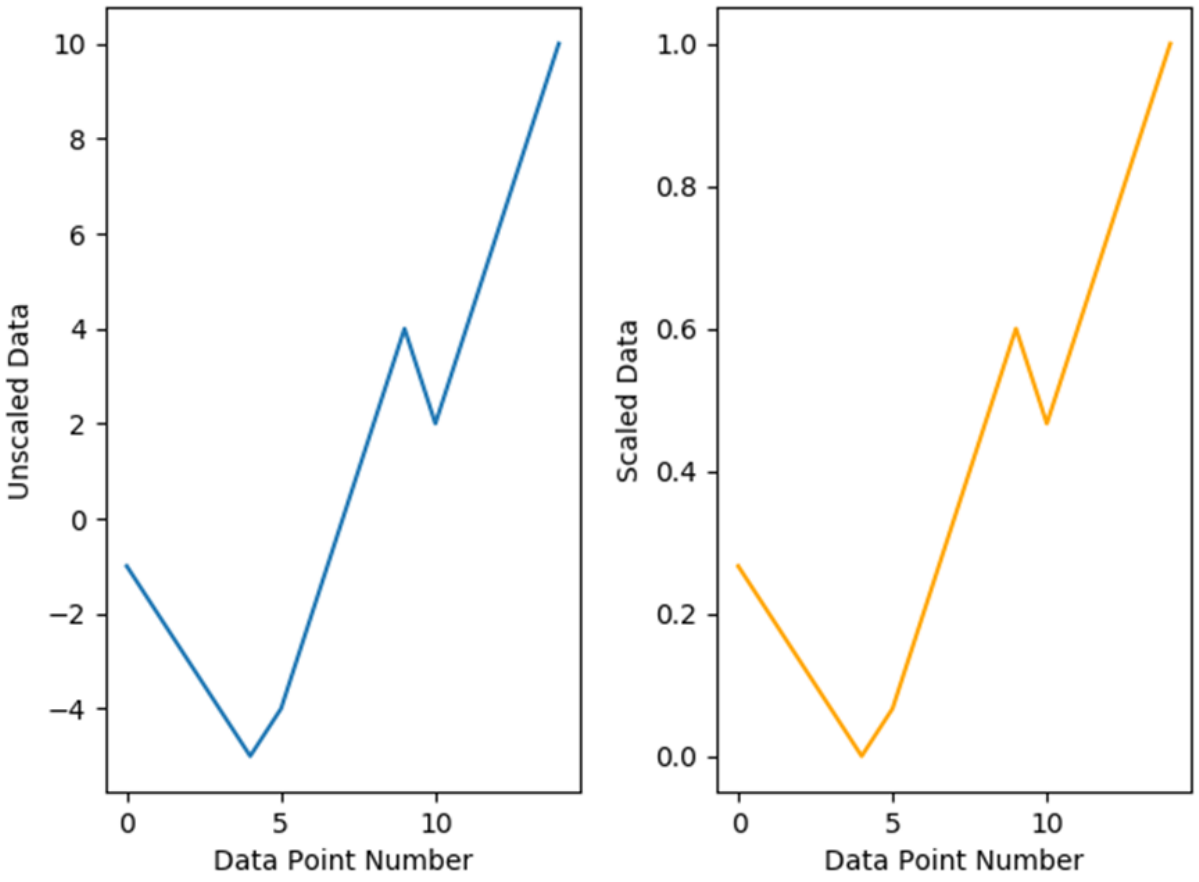


Figure 15: Comparison of Unscaled Data and Min Max Scaled Data

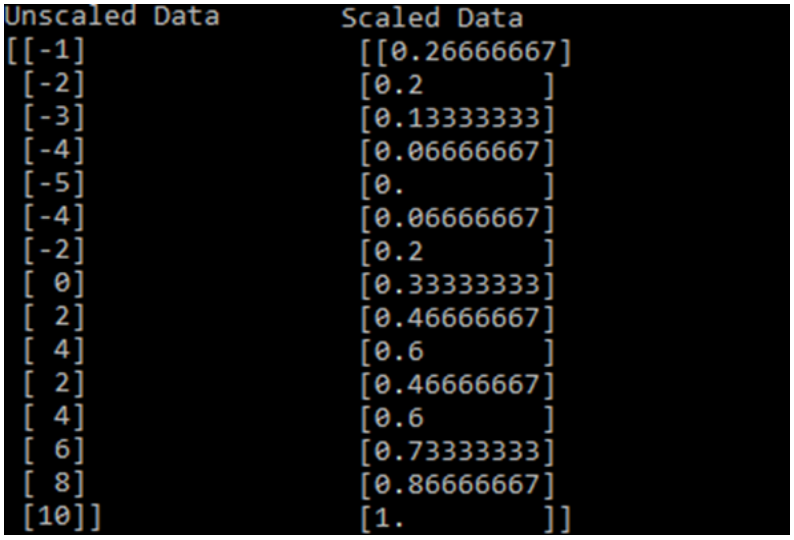


Figure 16: Example of Min Max Scaled Data

2.3.6 Normalization

The normalization method samples each feature independently to the unit norm. The scikit-learn method for normalization is .Normalizer(). The user is able to specify what type of regularization is used for the normalization, either L1 or L2. L1 is the sum of the regularization weights and L2 is the sum of the squares of the regularization weights. L1 typically is more robust and less efficient. L2 while less robust, will always be more efficient than L1. L1 and L2 are often referred to as least absolute deviations and least squares error, respectively. The default setting for this method is L2. Normalization is done using the following equation:

$$L2NormalizedValueofX = \frac{X}{\sqrt{\sum_{i=1}^{n} X_i^2}}$$

Normalizing data can help make the dataset less sensitive to the magnitude of the features and prevent bias. Also, data normalization is needed if a method such as Gaussian naïve Bayes is used, as these methods depend on data that is distributed normally. Finally, normalization can help with the convergence of the machine learning algorithm.

Figure 17 shows a comparison of normalized data and unscaled data. Figure 18 shows the output of the normalization method.

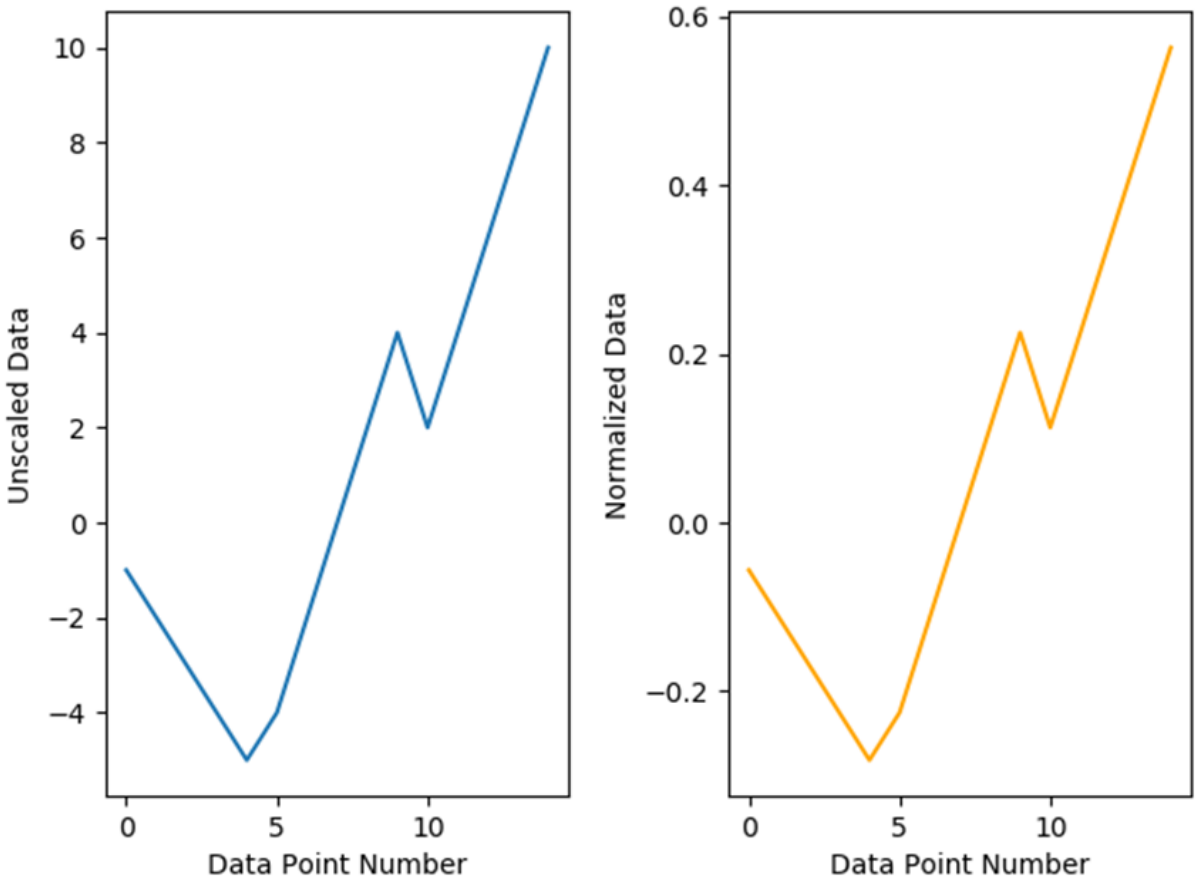


Figure 17: Comparison of Unscaled Data and Normalized Data

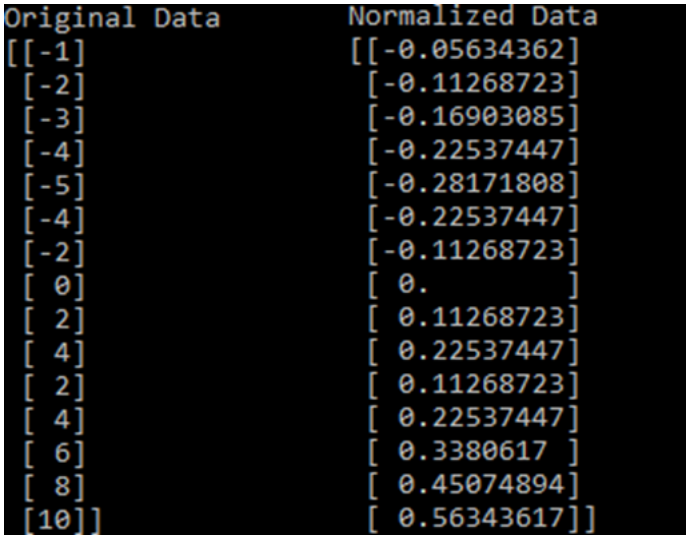


Figure 18: Example Normalized Preprocessed Data

2.3.7 Radial Basis Function Sampling

This project also make use of feature reduction techniques, to try and only use the components of each feature that are actually helpful for model training. One of these techniques is Radial Basis Function (RBF) sampling. RBF sampling is a form of Random Fourier Features, more commonly referred to as Random Kitchen Sinks. This technique is intended to replace weight minimization with randomization in order to improve the classification. The RBF sampler maps a kernel using a Monte Carlo approximation. This is used in kernel-based machine learning algorithms, such as k-nearest neighbors and k-means, but is more widely used in neural networks and support vector learning applications. RBF sampling is computationally less expensive than other kernel mapping techniques, such as the Nystroem Method, but can be less accurate. Due to this, RBF Sampling is better used in cases where there are clearer differences between classes[54].

In scikit-learn, the RBF Sampling function is .RBFSampler(). The user can specify the parameters of the kernel, gamma, as well as the dimensionality of the components. As with other Monte Carlo approximations, the more components used the better the accuracy, but more computation time is needed. A random number seed may be specified in

order to replicate the results. Once the RBFSampler parameters are set, the data must be fit to perform the Monte Carlo analysis and then transformed to model the kernel map.

2.3.8 Feature Agglomeration

Feature Agglomeration is a form a Hierarchical clustering, sometimes referred to as agglomerative clustering. This process takes in the data and groups similar data into a predefined number of groups or clusters. This technique is used as the basis for some unsupervised learning algorithms, but it has applications in supervised methods as well. In supervised learning this method can be used to aid in feature reduction for complex datasets to help deal with models that are overfit [55].

The scikit-learn method for Feature Agglomeration, .FeatureAllgomeration(), merges features together in order to reduce the number features used in a machine learning algorithm. The user has the ability to enter several parameters to better tune the reductions to the needs of the analysis. The first of these is the number of clusters that the methods will create. The default for this method is 2, as this is typically used to determine if it is appropriate to merge 2 features together to reduce overfit. Another key parameter for this method is the ability to change the linkage criteria of the method. This is called affinity. The linkage criteria available are: ward, single, complete and average. The ward criterion looks to minimize the variance between the features being merged. The average criterion uses the average distance of the features for linkage. The single and complete criteria uses the minimum and maximum distances respectively between the features for linkage. By default and for this project, the ward criterion was used. Finally, the user can define the affinity, the type of distance, applied to the criterion of the method. This is the metric that is used for linkage calculations. The options for affinity include: euclidean, l1,l2, manhattan, cosine or precomputed. Since this project uses the ward criteria, the euclidean metric must be used [56].

2.3.9 Principal Component Analysis

Principal Component Analysis (PCA) is a statistics-based technique that is designed to reduce the number of dimensions a dataset contains. PCA utilizes orthogonal transformations to combine correlated variables, using basic statistics and linear algebra techniques to identify patterns within data rather than using visualization. Once patterns are found within a dataset, it is then possible to reduce the number of features in that dataset based on the PCA results. This helps reduce model overfit and produces better machine learning models. PCA is also used in data compression applications as the original data can be recovered later if needed. PCA works by first removing the mean of each of the datasets features by subtracting the mean value of each feature by the individual points that correspond to that feature. A covariance matrix is then created; the size of this square matrix is equal to the number of features present in the dataset. The eigenvectors and eigenvalues of the covariance matrix are then calculated, the length of this eigenvectors is equal to 1. The eigenvalues are then sorted from highest to lowest, which shows the significance of each component. Eigenvalues with small significance can be removed and while some information is lost, that information is small and has less impact on the model while reducing model over fit. Using the eigenvalues that were kept, a new feature matrix is created. This matrix is then transposed and multiplied by the transpose of the mean adjusted data. This will give the new dataset with only the higher significant features left[57]. The user typically specifies the amount of variance that is acceptable to lose in the reduction. This technique is also useful for reducing statistical noise.

The scikit-learn function for PCA is .PCA(). The user is able to specify the amount of explained variance acceptable to lose for the creation of the covariance matrix, as well as specify an empirical mean, if needed. The function will detect the number of features in the dataset. It should be noted that PCA requires enough memory to fit all of the data present in the dataset. This can be a problem for very large datasets. In these cases PCA

can be performed in increments using the IncrementalPCA function[58].

2.3.10 Family Wise Error Feature Rate Selection

Model overfit due to an abundance of features is a major consideration when creating a machine learning model. To help manage this, the TPOT dictionary also included three feature reduction preprocessing techniques. These attempt to remove features that are non-informative from the model. The first of of these is Family Wise Error Feature Rate Selection (FWER).

The FWER method is an univariate statistical approach used in hypothesis testing. FWER is the probability of at least one false positive, Type 1 error, in a group of hypothesis test. It is calculated by taking the probability value (p-value) for a set of tests and rejecting hypothesis that fail a specified test. A common test for this rejection is the Bonferroni test[59]. This test rejects p-values based on the following expression:

$$Reject if p_i \geq \frac{\alpha}{h}$$

Where: alpha is the specified criteria for the hypothesis test and h is the number of hypothesis tested.

The scikit-learn method for FWER is .SelectFwe. This method uses a statistical approach in order to calculate the p-value, such as Chi-Square or F-value. The F-value is used by default. Also, an alpha may be specified by the use. The default alpha is 0.05.

Figure 19 and 20 show an FWER example from the scikit-learn documentation [32]. This example loads the wine data from the scikit-learn repository. This set has 178 samples and 13 features. The FWER method used Chi-Square to find p-values and a 0.01 alpha was selected for the evaluation criteria. Five features from the dataset failed the FWER test and were removed from the dataset.

```
from sklearn.datasets import load_wine
from sklearn.feature_selection import SelectFwe
from sklearn.feature_selection import chi2
wine = load_wine()
X, y = wine.data, wine.target
print('The Wine Dataset original size is:')
print(X.shape)
X_new = SelectFwe(chi2, alpha=0.01).fit_transform(X, y)
print('After applying the FWE method the wine dataset is now size:')
print(X_new.shape)
```

Figure 19: Example Code of Family Wise Error Rate Feature Selection

```
The Wine Dataset original size is:
(178, 13)
After applying the FWE method the wine dataset is now size:
(178, 8)
```

Figure 20: Sample Output of Family Wise Error Rate Feature Selection

2.3.11 Select Percentile

The select percentile method, similar to FWER, is another univariate selection method that uses a statistical test to determine which features should be removed from the dataset. The key difference between the select percentile method and FWER is that instead of specifying an alpha for the p-value rejection criteria, the user inputs a percentile value. P-values are either rejected or accepted, based on the scores when compared to that percentile. The scikit-learn function for the select percentile technique is .SelectPercentile(). Similar to the FWER function, the user selects the method for determining the p-values of the desired feature and the percentile for the p-value evaluation. This project uses the default F-value method for determining the p-values and a 10 percentile for the evaluation criteria.

2.3.12 Variance Threshold Selection

The variance threshold technique examines the features and does not factor the model outcomes, unlike univariate methods such as FWER. As such, variance threshold is an ideal feature selection method for unsupervised learning. Variance threshold selection calculates the variance of the individual features and removes those that do not meet the user specified requirements. The scikit-learn function for variance threshold is .VarianceThreshold(). This function requires the user inputs a variance for the evaluation criteria. The default value of this is 0, which removes features that have the same value in all samples. This was the method used for this project. Figure 21 from the scikit-learn documentation shows a code example of the technique being applied. This example cre-

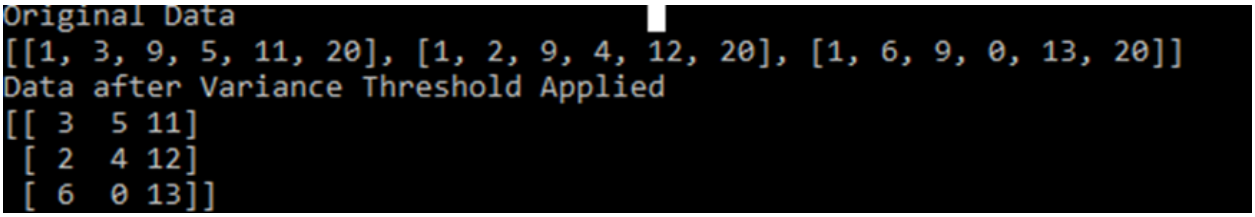


Figure 21: Example of Variance Threshold Feature Selection

ates a Python list with the 1st and 4th feature, 0 and 3, repeated in each row. The default variance of 0 is used for the feature selection. Once the method is applied, the repeated features have been removed from the dataset, leaving only the non-zero variance data in the set.

2.4 Machine Learning Models

When dealing with machine learning there is no rule of thumb to determine if one type of model will preform better than another model. This can only be determined by training and testing different models. TPOT can train and test several models and determine which produces the optimal model. However, it is useful to see how other models perform and compare to the optimal model. To do this, a custom TPOT dictionary can be used to limit the types of models trained in the process. In this phase of the project, six different models were trained using different machine learning techniques. These are: Naïve Bayes with a Bernoulli, Gaussian and multinominal distributions, logistic regression, decision tree classification and k-nearest neighbors. Table 2 summarizes these techniques. This chapter will go over each of these type of models and explain the advantages and disadvantages of each.

Table 2: Machine Learning Models Trained Using TPOT for Initial Experiment

Gaussian Naïve Bayes	Bernoulli Naïve Bayes	Multinomial Naïve Bayes
K-nearest Neighbors	Decision Tree Classification	Logistic Regression

2.4.1 Naïve Bayes Classification

Naïve Bayes classification is a supervised machine learning algorithm that relies on probabilities. This method is based on Bayes' theorem and assumes statistical independence between two data points. Naïve Bayes classification is used in a variety of industries, i.e. the medical industry and spam email detection. Due to the probabilistic nature of naïve Bayes classification, there are several different models that can be created based on distributions. This project uses three different naïve Bayes models: Gaussian, Bernoulli and multinomial. Detailed explanations on each will be given below[60].

2.4.1.1 Bayes Theorm Bayes' theorem when applied for naïve Bayes classification is given by the following equation:

$$P(Y|X_1,...,X_n) = \frac{(P(Y) * P(X_1,...,X_n|Y))}{P(X_1,....,X_n)}$$

Where:

P(Y) represents the probability of event Y occurring

$P(X_1,....,X_n)$ represents the probability of the X events occurring

$P(X_1,...,X_n \mid Y)$ represents the probability of the X events occurring given

$P(Y|X_1,...,X_n)$ represents the probability of event Y occurring given the X events It is important to note that naïve Bayes always assumes statistical independence, as such the relationship between the event Y and the X events can be simplified to the following equation:

$$P(Y|X_1,...,X_n) = \frac{(P(Y)\prod_{i=1}^{n} P(X_i|Y)}{P(X_1,...,X_n|Y)}$$

Under these assumptions, the Maximum A Posteriori (MAP) technique can be used to determine P(Y) and $P(X_i Y)$. In this case, the MAP technique estimates P(Y) based on the mode of $P(X_i|Y)$. The key difference between the several naïve Bayes techniques is how $P(X_i| Y)$ is calculated.

2.4.1.2 Naïve Bayes Advantages & Disadvantages Naïve Bayes classification models have several advantages over other supervised classification techniques. The first of these is that the time required to calculate the model is less than that of other techniques, such as k-nearest neighbors. Also, naïve Bayes techniques can be performed using less test data than other techniques, due to the conditional independence assumption of the technique. Finally, again due to the conditional independence assumption, the technique is less likely to suffer from overfitting due to a high number of features in

the dataset. This is because feature distributions are decoupled, allowing each feature distribution to be estimated as a single distribution. There are drawbacks to this technique though. Statistical independence is not common in the real world. As such, this technique is poor at creating estimates and is not a useful tool in regression analysis. Still, real-world applications have proven it to be an effective classification tool, as the dependencies between features tend to cancel out in the classification process[61].

2.4.1.3 Gaussian Naïve Bayes The first naïve Bayes method used for this project was the Gaussian classification technique. As the name suggests, this method assumes that the probabilities of features are Gaussian, or a standard normal distribution. Gaussian distributions are symmetrical and only have a single peak. The Gaussian naïve Bayes technique uses the following equation to calculate the probability of the set events Xi given event Y:

$$P(X_i|Y) = \frac{1}{\sqrt{2\sigma_y^2}} e^{\frac{-X_i-\mu_Y}{2\sigma_Y^2}}$$

Where:

μ_y is the mean of event Y

σ_y is the standard deviation of event Y.

2.4.1.4 Bernoulli Naïve Bayes The Bernoulli naïve Bayes method assumes the data follows a multivariate Bernoulli distribution. A Bernoulli distribution assumes that the values are Boolean, either 0 or 1. Due to this, the values entered into the model must be converted into this format. The probability of the set of events X_i given event Y, is found using the following equation:

$$P(X_i|Y) = P(i|y)x_i + (1 - P(i|y)(1 - x_i)$$

It should be noted that this method penalizes the score of the model if the feature i does not occur for a given Y.

2.4.1.5 Multinomial Naïve Bayes The final naïve Bayes method used for this project is the multinomial classification technique. Multinomial distribution is a generalized form of the binomial distribution. Unlike a binomial distribution, a multinomial distribution can have values other than 0 or 1. This is typically used to determine the probability of a series of mutually exclusive events occurring at a given time. A key difference between the multinomial and Bernoulli methods is that the multinomial technique does not penalize the score of the model if a feature does not occur within a given data point. Probability of a set of events X_i given event Y is calculated using the following equation:

$$P(X_i|Y) = \sigma_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Where:

N_{yi} is the number of times a features appears in the training set

N_y is the number of times a features occurs in both the testing and training sets

α: is a smoothing factor to prevent the non-occurrence of a feature from penalizing the model

2.4.1.6 Naïve Bayes Classification Example Unlike the k-nearest neighbor, naïve Bayes classification is difficult to visualize as it makes use of probabilities, rather than distance. As such, this example is provided in order to demonstrate how a naïve Bayes classification works in real-world applications. Perhaps, the most common application of naïve Bayes classification is for spam/junk email detection. In this type of classification there are only two possible outcomes: the email is either spam or it not spam. As naïve Bayes is a supervised method, the first step is to have a training set of email data that is already classified as either spam or not-spam.

The key with naïve Bayes classification is that the outcome probabilities must be calculated using the dataset features. The features of this type of dataset would be the words inside the email. Using the training set data, the probability that if an email contains a certain word that the email is spam is calculated. Since there are only two outcomes, the probability that given a certain word that the email is not spam is simply 1 minus the probability that given the word an email is spam. Typically, common words such as 'it' will be assigned neutral probabilities, while other keywords are assigned higher probabilities. Next, the probability of a general email being spam or not being spam is determined. This can be done using either the training dataset, to better tailor the model to a specific email account, or the probability can be assigned using outside data and assumptions [62].

Once all of the probabilities have been determined, the probabilities of an email being spam given a certain word, can be calculated using Bayes' theorem. This process is repeated for every word in the email. Once probabilities have been done for every word in the email, the probabilities are then combined, so the probability of the email being spam given the set of words can be determined. This is done using the following equation:

$$P(S|W) = \frac{P_1 * P_2 * \ldots P_N}{P_1 * P_2 * \ldots P_N + (1-P_1)(1-P_2)...(1-P_N)}$$

Where:

P(S | W) is the probability of the email being spam given the set or words

P_N are the probabilities of the email being spam given an individual word

Using this total probability, the system determines whether to classify the email as spam or not-spam, based on user preferences.

2.4.2 K-Nearest Neighbors

K-nearest neighbors is a supervised machine learning algorithm used in both, classification and regression models. This method is non-parametric, meaning that the algorithm does not rely on a set number of parameters and can be flexible depending on the situation. K-nearest neighbors works by using a user-defined constant integer known as k. K is the number of nearest neighbors that the algorithm looks for in the classification. A majority vote of the nearest neighbors is used to determine which class a data point belongs to. This means that the testing data point will be classified according to which training data it is closest to, in the feature space. The size of k must be balanced when using this method. If k is too small, the model may be overfit. If k is too large, the possibility of an under-fit model that leaves out important details increases[63].

The k-nearest neighbors algorithm also requires a distance function in order to calculate the distance between a given testing data point and the different training data points. This algorithm typically uses Euclidean distance, also known as the straight line distance between two points in Euclidean space. This is very similar to the distance formula used in basic algebra and geometry. It should be noted, that other distance metrics such as the Manhattan Distance, can be used instead. This project makes use of Euclidean distance. The Euclidean distance is found using the following equation:

$$D(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{|x_i|^2 + |x_j|^2 - 2x_i^T x_j}$$

Where:

$D(x_i, x_j)$ is the distance between the 2 points

$|x_i|^2$ and $|x_j|^2$ are norm of the respective points

$2x_i^T x_j$ is the dot project between the two points

The k-nearest neighbors algorithm works best when the data points are scaled to bal-

ance the magnitude of different features and are normalized. K-nearest neighbors is simple in nature and easy to visualize, especially in datasets with fewer features. Also, with more data and a large k, the algorithm can produce very accurate results. Unfortunately, there are disadvantages associated with using k-nearest neighbors. The first is that it can be computationally expensive, especially with larger datasets. Also, it can be sensitive to statistical noise from features. Finally, the model can suffer if too many features are used[64].

2.4.3 Logistic Regression

Logistic regression is a supervised machine learning algorithm designed to deal with complex scenarios. While its name suggests this method is a regression method, it is actually a classification method. Similar to the naïve Bayes method, logistic regression uses probabilities to predict to what class a set of features belongs to, i.e. the probability of Y given X, or a set Xi. Outputs can be Boolean, multinomial or for cases where more than one class exists, One vs. Rest. An important difference between the two methods is that naïve Bayes assumes the features are statistically independent, while logistic regression does not. This results in the naïve Bayes model having more bias but less variance when compared to a logistic regression model. The decision of which model is best depends on the data used in creating the two models. Typically, logistic regression is preferred when the data has a large number of features, while naïve Bayes works better with less complex data. Logistic regression takes the different outcome probabilities of a given data point and models them using a logistic function[65]. This project makes use of several different categories for classification. The One vs. Rest method of logistic regression will be used to create the classification model. One vs. Rest logistic regression uses the following equation:

$$P(Y_k|X) = \frac{1}{1+\sum_{j=1}^{k=1} e^{w_{jo}+\sum_{i=1}^{n} w_{ji}X_i}}$$

Where:

$P(Y_{k|X})$ is the probability of Y belonging to class k given X

w_{jo} and w_{ji} are the weights associated with class j Scikit-learn has the ability to optimize this equation by modifying the weights with user specified input[66]. This can be done using three methods: L1, L2 or Elastic Net regularization. L1 solves issues with weight optimization given by the following equation:

$$L1 = min_{wc}|w| + C\sum_{i=1}^{n} log(e^{-y_i(X_i^T w+c)}) + 1$$

L2 minimizes the cost function of the weights using the following equation:

$$L2 = min_{wc}\frac{1}{2}w^T w + C\sum_{i=1}^{n} log(e^{-y_i(X_i^T w+c)}) + 1$$

and Elastic Net regularization is used when there are issues with both, cost and optimization using the following equation:

$$min_{wc}\frac{1-p}{2}w^T w + p|w| + C\sum_{i=1}^{n} log(e^{-y_i(X_i^T w+c)}) + 1$$

2.4.4 Decision Tree Classification

Decision tree analysis is a non-parametric, supervised learning method that can be used in both classification and predictive regression. Decision tree analysis, unlike other methods, such as k-nearest neighbors and naïve Bayes, is designed to deal with statistical noise that can deter a model's performance. Common applications for decision tree analysis include: credit and loan assessment, medical diagnostics and performance evaluation/prediction.

The top of a decision tree is the outcome of the analysis. Decision tree analysis begins by taking instances from the data provided. These are typically the different classes of the data provided to the model. Next, a data point is tested, typically using a true or

false evaluation, though it is possible to use non-boolean responses for testing, if appropriate. Depending on the outcome, further testing may occur or the model may have enough information to make a determination and classify a data point. If more testing is needed, the model will continue to evaluate the data point provided until a determination can be made or there are no more evaluation metrics to test. A good analogy for a decision tree classification would be a personality test, such as the Myers-Briggs Type Indicator. In this survey, the respondent is asked several questions and at the end of the test, the user is put in a class based on the responses to the questions. In a decision tree classification, the features provided in a data point are the responses to the questions at each root of the tree[67]. Decision trees have several advantages over other machine learning methods. Unlike logistic regression or naïve Bayes methods, decision trees are easy to visualize and conceptually simple. Decision trees can work with both, qualitative and quantitative data, and data does not need to be of the same type. Also, decision trees are better able to deal with missing values and a classification can still be done if some features are missing. This reduces the amount of data preparation that needs to be done in order to perform the analysis. Finally, decision trees are able to handle multiple output models, which allow the method to apply more complex problems. Still, there are some disadvantages associated with decision trees. The order of the evaluations in the tree is extremely important. Some orders may filter out critical data and lead to inaccurate results and misclassifications. This can be managed by creating several different trees to determine which order best fits the model. This can be computationally expensive and time consuming. A better method of dealing with this would be to calculate an evaluation's entropy or Gini Impurity. This will be explained below. Another issue with decision trees is that models can become overly complex and are more prone to overfit, if too many features are present. Data preprocessing and selection methods, such as PCA, as well as tree pruning can be used to help manage this issue.

2.4.4.1 Entropy In order for decision trees to be most effective, the manner in which the tree is built is critical. Ideally, the most valuable features would be near the top of the tree allowing for quicker determinations. One approach is the use of entropy. In this context, entropy is a measure of the purity/impurity of the data samples and can be used to calculate the information gain from each features used in the tree's construction. This can be calculated using the following equation:

$$S = \sum plog_2 p$$

Where:
S is the Entropy of the collection of data
P is the mass probability function of the evaluation If the responses to the evaluation are Boolean, the equation can be expressed as the combination of the negative responses and the positive responses with the following equation:

$$S = -plog_2 p - plog_2 p$$

Evaluations with higher entropy are considered more insightful and are prioritized earlier in the tree, while those with lower entropy values are placed lower in the tree. This allows the model to determine the sequence that yields the most gain[68]. This approach is known as a 'greedy' algorithm, where the algorithm searches for the optimal result rather than the best. Decision Trees trained using entropy with sklearn are based on the Iterative Dichotomiser 3 (ID3) algorithm.

2.4.4.2 Gini Impurity Another approach to building a decision tree is to use Gini Impurity. Gini Impurity uses the features provided to calculate the probability of a misclassification. The probability is calculated using the following equation:

$$G = \sum p(1-p)$$

Where:

G is the probability of a misclassification
p is the probability of selecting a given data point within a dataset

Once the probabilities for each feature have been calculated, the features with the lowest Gini Impurity should be placed at the top of the tree. This is the approach that will be used for all decision tree models trained in this project. Sklearn's Gini Impurity decision tree algorithms are based off of the Classification and Regression Tree (CART) algorithm.

2.5 Model Validation

Once a machine learning model has been trained and tested, it is necessary to validate the model to determine how well the model performed. Regression models use metrics like Mean Absolute Error and Relative Square Error to evaluate the model's performance. Classification models such as those trained during this project can make use of a variety of metrics for performance, such as accuracy and recall. This section will go over the different evaluation metrics used on this project.

2.5.1 Accuracy

Accuracy is the most common validation measurement used to assess a model's performance. The accuracy of the model is simply the number of correct classifications divided by the total number of test classifications performed[69]. Scikit-learn uses the .accuracy_score() method to measure a model's accuracy using the following expression:

$$accuracy(y, y_{bar}) = \frac{1}{n_samples} \sum_{i=1}^{n_{samples}-1} (y_{bar} = y_i))$$

2.5.2 Precision

In addition to accuracy, other validation measurements are needed when evaluating a classification model. Relying only on accuracy measurements would only tell the user that a misclassification had occurred. Using other validation measurements allows for further exploration of a model's mistake and the ability to identify potential issues with a model. Measuring a model's precision allows the user to see the ratio of true positives to predicted positives. This shows how many false positives, or Type I errors, occurred in the model's testing. Scikit-learn uses the .precision_score() method to calculate a model's precision using the following equation:

$$Precison = \frac{Number of True Positives}{Number of False Positive + Number of True Positives}$$

2.5.3 Recall

The next validation measurement performed for this project is recall. Recall is a measurement of a model's ability to classify positive samples. This is referred to as sensitivity. Recall allows the user to evaluate a model's false negative error, or Type II error. Scikit-learn uses the .recall_score method to calculate a model's recall using the following equation:

$$Recall = \frac{Number of True Positives}{Number of False Negatives + Number of True Positives}$$

2.5.4 F1 Score

In classifying reactor transients, false positive errors potentially result in action that is costly and time consuming. False negatives can result in potential reactor damage and

events hazardous to the public. Both of these must be considered when evaluating models. As such, it is necessary to have a validation measurement that balances Type 1 and Type II error. This measurement is known as the F1 score. The F1 score measures a weighted average between precision and recall. Scikit-learn uses the .f1_score() function to calculate a model's F1 score using the following equation:

$$F1 = 2 * \frac{Precision * Recall}{/} Precision + Recall$$

It should be noted the the F1 score can take different forms if it is determined that the model evaluation should put a higher emphasis on either recall or precision. This project will use the balanced form of the F1 score.

2.5.5 Confusion Matrix

In dealing with percentages it is important to consider the context of the situation. In this phase of the project, over 15,000 samples are being tested by the model. In these cases a 1% change in accuracy or precision would affect 150 samples. As such, it is necessary to use a method that tells the exact number of Type 1 and Type II errors that occurred and between which transients these occurred. To do this a confusion matrix can be created. A confusion matrix shows the true positives for each classification down the diagonal of the matrix. The false positives are shown in the columns and the false negatives are in the rows. This allows not only for the exact number and type of errors to be shown, but also where they occurred. This can provide insight into where and why a classification model is having issues. These results can also be used to determine the model's ability to classify specific classes within the dataset. Confusion matrices will be generated for models trained in this project to diagnose reactor transients.

2.6 Generic Pressurized Water Reactor Simulator

Due to the high cost of nuclear reactors and the risk associated with a reactor accident, it is unfeasible to collect data for model training using actual reactor transients. Instead, this project relies on synthetic data from a reactor simulator. Data used for this experiment was collected using the Generic Pressurized Water Reactor (GPWR) simulator located in the Audio Visual Laboratory at theCenter for Advance Energy Studies (CAES) in Idaho Falls, Idaho. The GPWR simulator was purchased by the University of Idaho, Idaho State University and INL from the Western Services Cooperation (WSC). This section will provide an overview on the GPWR simulator and it's capabilities.

GPWR emulates the behavior of a 'generic' pressurized water reactor (PWR). The thermal output is rated at 4000 MWth/1400 MWe. Although the simulator does not directly incorporate the design of any specific PWR, it is based off a KEPCO APR1400, a South Korean design[70]. The reactor systems include a single high-pressure turbine, three low-pressure turbines, and a configuration that includes two loops, four coolant pumps and two steam generators. It also provides simulation of a switch yard, transmission lines, and two loads; representative of cities. This simulator fits best under IAEA's basic principle simulator category, as it does not mimic a specific reactor. Typical use of WCS simulators includes reactor operator training at several nuclear sites around the world and research that involves analyzing reactor behavior. Figure 22 shows the simulator setup at the CAES Audio Visual Laboratory.

2.6.1 Simulator Capabilities

The GPWR has a number of different pre-programmed operating conditions. The simulated conditions include reactor core life, power level, and operational state. The user is able to switch between several different control panels to control a variety of the plants components, such as pumps and breakers. This allows the user to be able to control functions, such as coolant flow to the reactor, power produced, etc. It should be noted that

Figure 22: GPRW Reactor Simulator at CAES

the simulator is able to replicate the behavior of all safety systems in the reactor including the containment under different accident events, as well as operating normally. For example, under certain conditions, the simulator will scram the reactor. The user may also select the individual components and designate them as malfunctioned. Possible malfunctions include: heat exchanger degradation, motor shearing/seizures, valve failure to open/close, etc. These events can be triggered by the user or programmed as part

of an accident scenario. Figure 23 shows an overview display that is used to navigate and control the different functions.

The GPWR displays critical parameters to the user. Many of these are those that an operator would see while running an actual reactor. There are 18 different parameters that are always displayed to the user. Figure 24 shows how this output is displayed to the user. The user is also able to see how long the simulator has been running, as well as pause and restart simulations. The simulation can be run in real-time, slowed down to 0.1 times normal speed or speed up to 10 times normal speed. This can be changed at any time during the simulation. In addition, the user has the ability to backtrack to a previous time in the simulation. The software will automatically save conditions every few minutes in order for the user to easily return to a previous state. These save states are usually over 100 MB in size. The simulator also allows for the user to switch between different interfaces in order to observe, manipulate and record the behavior of components that are not shown on the reactor interface page. This is done using a navigation panel which allows the user to select a more detailed interface of a specific component. This is shown in Figure 22.

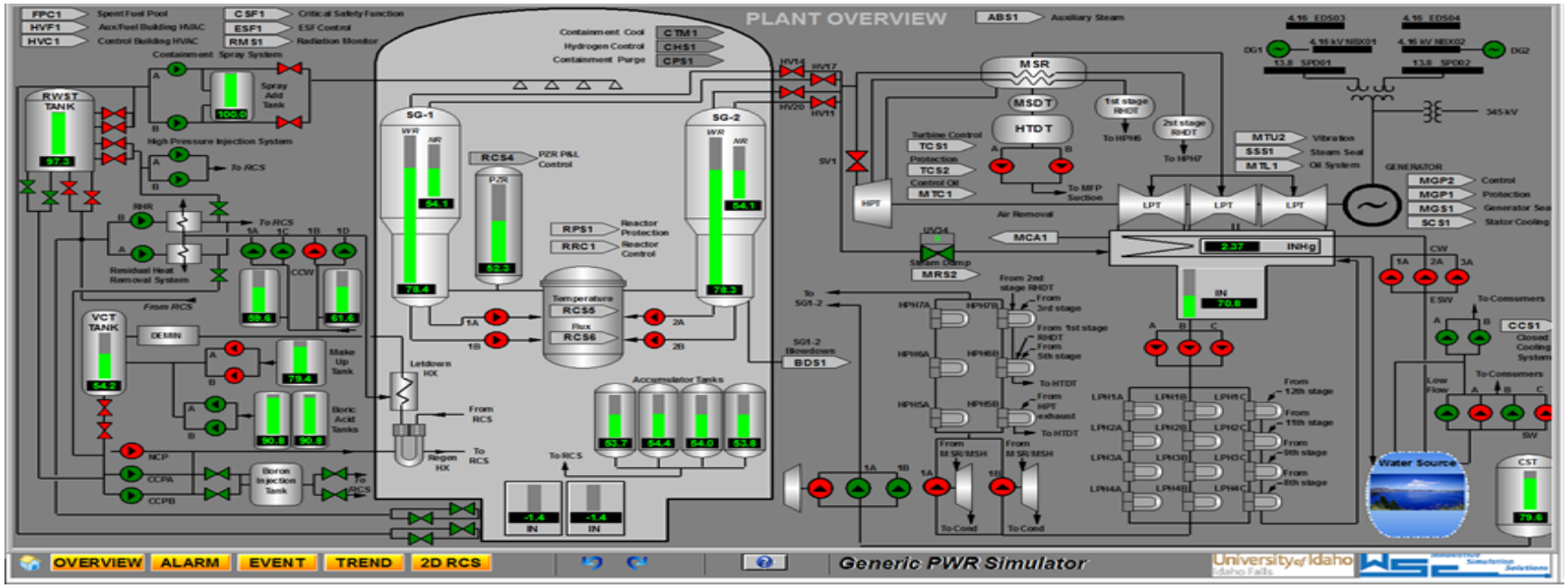
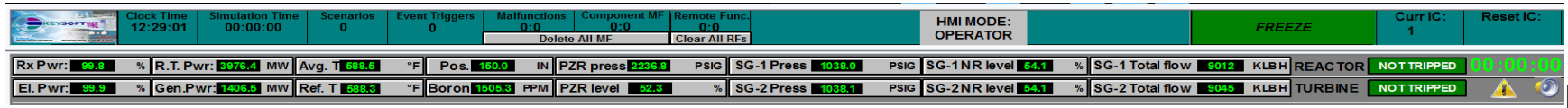


Figure 23: GPRW Interface



[h!]

Figure 24: GPWR Simulator Setup

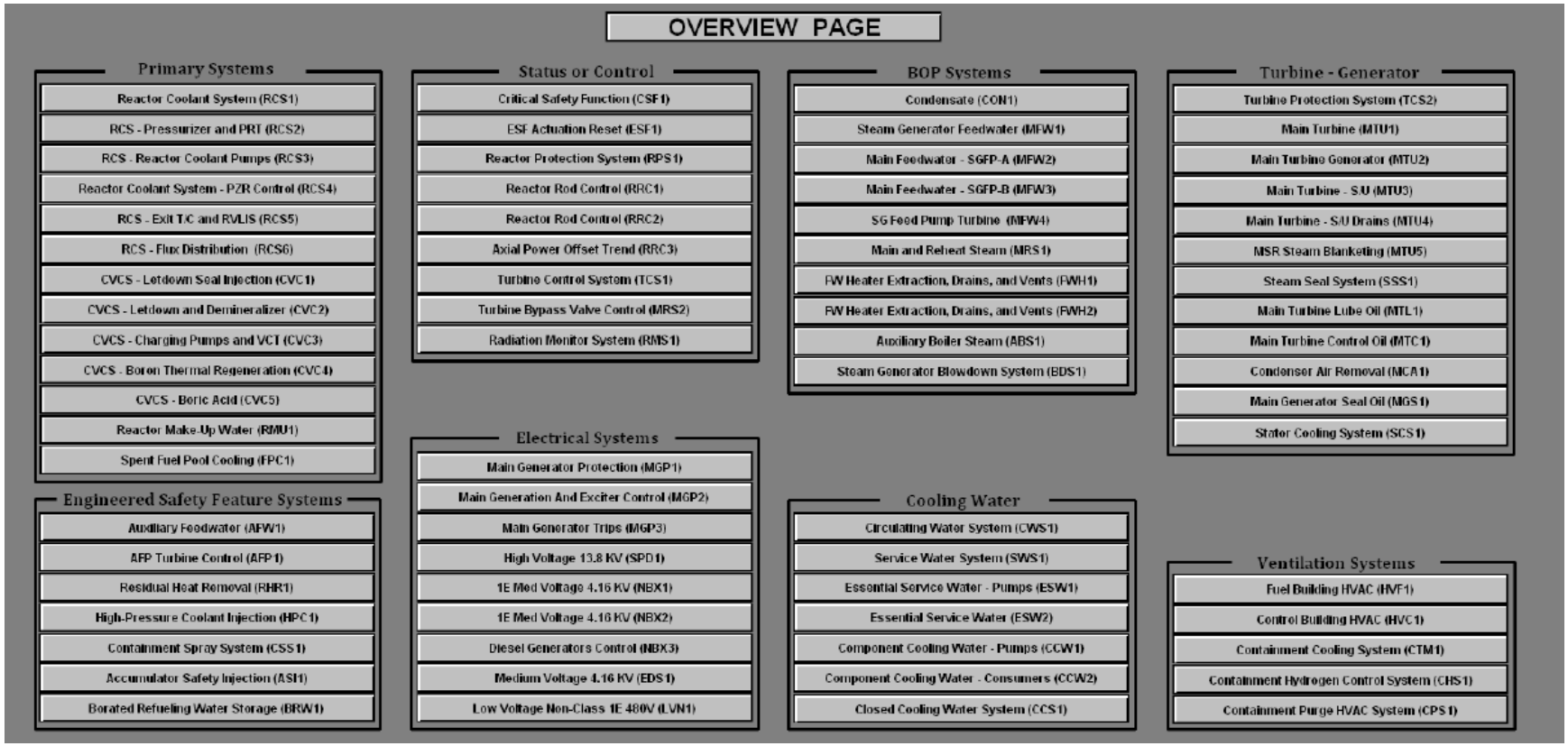


Figure 25: GPWR Simulator Overview Panel

2.6.2 Verification & Validation

One of the changes that came out of the TMI accident was the expanded use of simulation in the nuclear industry. In order for a simulator to be effective, it must be able to represent the systems being simulated accurately to what would occur in the real world. In the United States, the NRC plays a large role in this. The NRC has developed a Verify & Validate (V&V) standard for approving software that will be used in nuclear simulations. This is covered under NRC Regulatory Guide 1.168.

Regulatory Guide 1.168 covers the basic outline for gaining approval for nuclear related software related to safety systems. This includes how such codes are updated and audited while they are in use. These guidelines are an adaptation IEEE Std 1012-1998, Standard for Software Verification and Validation. Some of the steps in the V&V process include independent quality assurance audits, security assessment, as well as providing guidelines for software validation. Well known software that has gone through this process includes: INL's Sapphire for systems analysis and SPAR models for reactor oversight[71].

In order to ensure that simulations performed by the GPWR are representative of real reactor behavior, the GPWR makes use of several validated reactor codes. Some examples include RELAP5 and MARS, developed by INL and the Korea Atomic Energy Research Institute respectively, for thermal hydraulics. The NESTLE code, developed by NCSU, for neutronics and the MELCOR code from Sandia National Lab for accident modeling were also used. This type of simulator is similar to those used by INL's Human Systems Simulation Lab. Table 3 summarizes many of the codes used in the GPWR

Table 3: Nuclear Software Codes using in GPWR

Software Code	Primary Developer	Regulator	Purpose in GPWR
RELAP5	North Carolina State University	NRC	Thermal-hydraulics
Nestle	Idaho National Lab	NRC	Neurotics
MARS	Korea Atomic Energy Research Institute	NRC	Thermal-hydraulics
MELCOR	Sandia National Lab	NRC	Accident Modeling

3 Developing Initial Machine Learning Models Using TPOT

The first goal of this project was to determine if TPOT could train models that could correctly identify a small number of transients in a reactor system. This chapter will go over the methodology used to perform this experiments, such as data collection and model training. Finally, this chapter will go over the results of the first phase of this study.

3.1 Methodology

3.1.1 Data Collection

The most critical step, as is the case with most data science projects, is to collect data. As mentioned, this project relies on synthetic data collected from the GPWR simulator. It was decided that for this project, the data gathered would consist of features that a reactor operator would likely have access to and be readily available. Thirty three features were chosen and programmed into the simulator data collector, including: reactor power output, steam generator temperature, flow and pressure, as well as reactor temperature. Table 4 summarizes the features that were collected for thee initial dataset. All of the features collected from the reactor simulator were quantitative. In order to see how the model would be impacted by changes in the reactor system over time, it was decided that several simulations would be conducted changing the initial conditions of the simulated system for each run. The first change to the system was the power output of the reactor. Three different conditions were used: full power, where the reactor is operating as to generate electricity; half power, where the reactor is being shut down and output is at approximately 50%; and low power where the reactor is critical and being prepared for startup, but power generation is between 0 and 1% capacity. The second initial condition changed for the reactor system involves the stage of the reactor's lifetime. Three different conditions were available for use: Beginning of Life (BOL), where the reactor is brand new; Middle of Life (MOL), where the reactor is close to 30 years old; and End of

Table 4: Features Collected from GPWR Simulator

Normalized Flux	RCS LVL Loop 1 WR
RCS LVL Loop 1 NR	Hot Leg 1 Temperature
Hot Leg 2 Temperature	Cold Leg 1B Temperature
Cold Leg 2B Temperature	Cold Leg 1A Temperature
Cold Leg 2A Temperature	RC Loop 1A Norm Flow
RC Loop- 2A Norm Flow	RC Loop 1B Norm Flow
RC Loop 2B Norm Flow	Containment Temperature
Pressurizer Surge Line Temperature	PORV Discharge Pressurizer Temperature
Containment Pressure	SG-1 NR Level
SG-2 NR Level	FW Flow to SG-1
FW Flow to SG-2	Pressurizer Pressure
MS Flow from SG-1 Line-1B	MS Flow from SG-1 Line-2B
SG-1 Pressure	SG-2 Pressure
Pressurizer Steam Temperature	Norm Pressurizer Level
Pressurizer Water Temperature	Generator Power
Average Temperature	MS Flow from SG-1 Line-1A
MS Flow from SG-1 Line-2A	

Life (EOL), where the reactor is close to decommission, approximately 60 years into its operating life. Using these two features, it was possible to collect data on nine different initial condition combinations while the reactor is functioning as intended. Each run was conducted for 1200 seconds and data was collected for each of the 33 measurable features every second during the run. Seconds are the smallest increment of time that can be used for data collection on the GPWR. Table 5 list the initial conditions used in this part of the study.

3.1.2 Transient Events

In addition to collecting data when the reactor is under normal operating conditions, it was also necessary to collect data with the system experiencing transient events. The NRC defines a transient event "A change in the reactor coolant system temperature, pressure, or both, attributed to a change in the reactor's power output."[72]. This includes accident conditions. For the purposes of this study, it should be assumed that all transient

Table 5: Initial Conditions used for Simulations

	Core Life	1 Power Level
Initial Condition 1	BOL	100%
Initial Condition 2	MOL	100%
Initial Condition 3	EOL	100%
Initial Condition 4	BOL	50%
Initial Condition 5	MOL	50%
Initial Condition 6	EOL	50%
Initial Condition 7	BOL	1% (Critical)
Initial Condition 8	MOL	1% (Critical)
Initial Condition 9	EOL	1% (Critical)

events are either anticipated operational occurrences or postulated accident events, as defined in 10 CFR Part 50. Four transient events were chosen for this part of the study. Simulations were ran using as many of the nine initial conditions as applicable for the each transient. This section will go into detail on the transient events simulated. The first transient event selected was a simultaneous trip of all feed water pumps. In this transient, the primary and auxiliary feed water pumps malfunction and cease operations. The breakers connected to these two pumps also trip. The transient was programmed to occur 20 seconds after the simulation began. A simulation under each of the nine different initial condition configurations was performed and data was collected for 600 seconds after the transient occurred. Under this transient, the runs that were performed at full power and half power scrammed the second the transient occurred. During the run that occurred at low power, no scram occurred during the simulation[73].

The next transient event that was used to collect data was a simultaneous closure of Main Steam Isolation Valves (MSIV). In this transient, a command signal is sent to all MSIVs after 20 seconds, switching the valves from the open position to the closed position. Data was collected for 600 seconds after the command signal was sent. Each of the nine different initial condition configurations was used to collect data on this transient. In this event, runs performed under full and half power experienced a scram 40

second after the simulation began, 20 seconds after the command signal was sent. Runs performed at low power did not scram during the simulation.

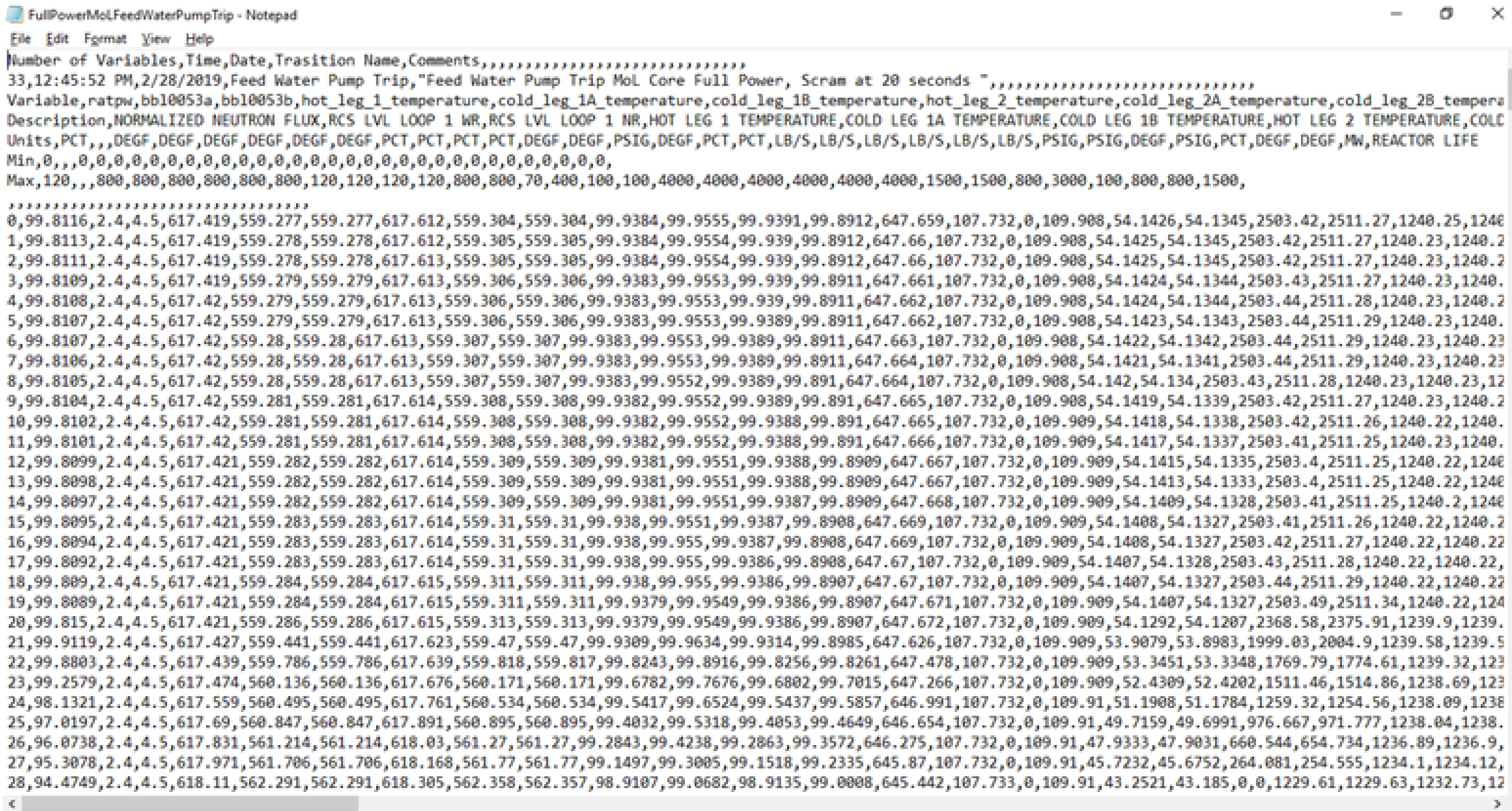
The third transient event used in this experiment was a maximum reactor coolant rupture combined with a complete LOOP. During this transient, a double ended guillotine break occurs within line 1A of the reactor coolant system. This is combined with a complete loss of electrical power to the plant. The transient occurred 20 seconds after the simulation began and data was collected for 600 seconds after the transient occurred. Nine separate runs were performed using the initial condition configurations. During this simulation, the reactor experienced a scram at all power levels used. The final transient event used to collect data was a rapid power change. In this transient, the reactor begins 1400 MWe full power and drops to 1050 MWe, approximately 75% of the plant's maximum power, before returning to 1400 MWe. Data was collected until the reactor reached full power, approximately 1000 seconds. Due to the nature of this transient, only the reactor core life initial conditions were changed and three runs were performed.

3.1.3 Dataset Preperation

After the completion of a run, the data was saved from the reactor simulator to a Comma Separated Values (CSV) file. In total, 39 different CSV files were generated and saved. Table 6 list the simulations that were performed for each. Again, all data gathered directly from the simulator was quantitative. After each run, the reactor core lifetime feature was added to each instant from the dataset from the run using either BOL, MOL or EOL. Also, the transient that occurred was added to each instant in the dataset. It should be noted, that the instances up to the 20 second mark were labeled as normal operations, as the transient had not yet occurred. These additions were done using Microsoft Excel. The datasets remained in a CSV format. Figure 26 shows a screenshot of one of the CSV files collected from the simulator.

Table 6: Simulations Run for Initial Dataset

Simulation 1	BOL	100%	Normal Operations
Simulation 2	MOL	100%	Normal Operations
Simulation 3	EOL	100%	Normal Operations
Simulation 4	BOL	50%	Normal Operations
Simulation 5	MOL	50%	Normal Operations
Simulation 6	EOL	50%	Normal Operations
Simulation 7	BOL	1%(Critical)	Normal Operations
Simulation 8	MOL	1%(Critical)	Normal Operations
Simulation 9	EOL	1%(Critical)	Normal Operations
Simulation 10	BOL	100%	Feedwater Pump Trip
Simulation 11	MOL	100%	Feedwater Pump Trip
Simulation 12	EOL	100%	Feedwater Pump Trip
Simulation 13	BOL	50%	Feedwater Pump Trip
Simulation 14	MOL	50%	Feedwater Pump Trip
Simulation 15	EOL	50%	Feedwater Pump Trip
Simulation 16	BOL	1%(Critical)	Feedwater Pump Trip
Simulation 17	MOL	1%(Critical)	Feedwater Pump Trip
Simulation 18	EOL	1%(Critical)	Feedwater Pump Trip
Simulation 19	BOL	100%	MSIV Closure
Simulation 20	MOL	100%	MSIV Closures
Simulation 21	EOL	100%	MSIV Closure
Simulation 22	BOL	50%	MSIV Closure
Simulation 23	MOL	50%	MSIV Closure
Simulation 24	EOL	50%	MSIV Closure
Simulation 25	BOL	1%(Critical)	MSIV Closures
Simulation 26	MOL	1%(Critical)	MSIV Closure
Simulation 27	EOL	1%(Critical)	MSIV Closure
Simulation 28	BOL	100%	LOCA-LOOP
Simulation 29	MOL	100%	LOCA-LOOP
Simulation 30	EOL	100%	LOCA-LOOP
Simulation 31	BOL	50%	LOCA-LOOP
Simulation 32	MOL	50%	LOCA-LOOP
Simulation 33	EOL	50%	LOCA-LOOP
Simulation 34	BOL	1%(Critical)	LOCA-LOOP
Simulation 35	MOL	1%(Critical)	LOCA-LOOP
Simulation 36	EOL	1%(Critical)	LOCA-LOOP
Simulation 37	BOL	100%	Rapid Power Change
Simulation 38	MOL	100%	Rapid Power Change
Simulation 39	EOL	100%	Rapid Power Change



```
FullPowerMoLFeedWaterPumpTrip - Notepad
File Edit Format View Help
Number of Variables,Time,Date,Trasition Name,Comments,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
33,12:45:52 PM,2/28/2019,Feed Water Pump Trip,"Feed Water Pump Trip MoL Core Full Power, Scram at 20 seconds ",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
Variable,ratpw,bbl0053a,bbl0053b,hot_leg_1_temperature,cold_leg_1A_temperature,cold_leg_1B_temperature,hot_leg_2_temperature,cold_leg_2A_temperature,cold_leg_2B_tempera
Description,NORMALIZED NEUTRON FLUX,RCS LVL LOOP 1 WR,RCS LVL LOOP 1 NR,HOT LEG 1 TEMPERATURE,COLD LEG 1A TEMPERATURE,COLD LEG 1B TEMPERATURE,HOT LEG 2 TEMPERATURE,COLD
Units,PCT,,,DEGF,DEGF,DEGF,DEGF,DEGF,DEGF,PCT,PCT,PCT,PCT,DEGF,DEGF,PSIG,DEGF,PCT,PCT,LB/S,LB/S,LB/S,LB/S,LB/S,LB/S,PSIG,PSIG,DEGF,PSIG,PCT,DEGF,DEGF,MW,REACTOR LIFE
Min,0,,,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
Max,120,,,800,800,800,800,800,800,120,120,120,120,800,800,70,400,100,100,4000,4000,4000,4000,4000,4000,1500,1500,800,3000,100,800,800,1500,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
0,99.8116,2.4,4.5,617.419,559.277,559.277,617.612,559.304,559.304,99.9384,99.9555,99.9391,99.8912,647.659,107.732,0,109.908,54.1426,54.1345,2503.42,2511.27,1240.25,1240
1,99.8113,2.4,4.5,617.419,559.278,559.278,617.612,559.305,559.305,99.9384,99.9554,99.939,99.8912,647.66,107.732,0,109.908,54.1425,54.1345,2503.42,2511.27,1240.23,1240.2
2,99.8111,2.4,4.5,617.419,559.278,559.278,617.613,559.305,559.305,99.9384,99.9554,99.939,99.8912,647.66,107.732,0,109.908,54.1425,54.1345,2503.42,2511.27,1240.23,1240.2
3,99.8109,2.4,4.5,617.419,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.939,99.8911,647.661,107.732,0,109.908,54.1424,54.1344,2503.43,2511.27,1240.23,1240.
4,99.8108,2.4,4.5,617.42,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.939,99.8911,647.662,107.732,0,109.908,54.1424,54.1344,2503.44,2511.28,1240.23,1240.2
5,99.8107,2.4,4.5,617.42,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.9389,99.8911,647.662,107.732,0,109.908,54.1423,54.1343,2503.44,2511.29,1240.23,1240.
6,99.8107,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9553,99.9389,99.8911,647.663,107.732,0,109.908,54.1422,54.1342,2503.44,2511.29,1240.23,1240.23
7,99.8106,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9553,99.9389,99.8911,647.664,107.732,0,109.908,54.1421,54.1341,2503.44,2511.29,1240.23,1240.23
8,99.8105,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9552,99.9389,99.891,647.664,107.732,0,109.908,54.142,54.134,2503.43,2511.28,1240.23,1240.23,12
9,99.8104,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9389,99.891,647.665,107.732,0,109.908,54.1419,54.1339,2503.42,2511.27,1240.23,1240.2
10,99.8102,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9388,99.891,647.665,107.732,0,109.909,54.1418,54.1338,2503.42,2511.26,1240.22,1240.
11,99.8101,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9388,99.891,647.666,107.732,0,109.909,54.1417,54.1337,2503.41,2511.25,1240.23,1240.
12,99.8099,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9388,99.8909,647.667,107.732,0,109.909,54.1415,54.1335,2503.4,2511.25,1240.22,1240
13,99.8098,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9388,99.8909,647.667,107.732,0,109.909,54.1413,54.1333,2503.4,2511.25,1240.22,1240
14,99.8097,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9387,99.8909,647.668,107.732,0,109.909,54.1409,54.1328,2503.41,2511.25,1240.2,1240
15,99.8095,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.9551,99.9387,99.8908,647.669,107.732,0,109.909,54.1408,54.1327,2503.41,2511.26,1240.22,1240.2
16,99.8094,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.955,99.9387,99.8908,647.669,107.732,0,109.909,54.1408,54.1327,2503.42,2511.27,1240.22,1240.22
17,99.8092,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.955,99.9386,99.8908,647.67,107.732,0,109.909,54.1407,54.1328,2503.43,2511.28,1240.22,1240.22,
18,99.809,2.4,4.5,617.421,559.284,559.284,617.615,559.311,559.311,99.938,99.955,99.9386,99.8907,647.67,107.732,0,109.909,54.1407,54.1327,2503.44,2511.29,1240.22,1240.22
19,99.8089,2.4,4.5,617.421,559.284,559.284,617.615,559.311,559.311,99.9379,99.9549,99.9386,99.8907,647.671,107.732,0,109.909,54.1407,54.1327,2503.49,2511.34,1240.22,124
20,99.815,2.4,4.5,617.421,559.286,559.286,617.615,559.313,559.313,99.9379,99.9549,99.9386,99.8907,647.672,107.732,0,109.909,54.1292,54.1207,2368.58,2375.91,1239.9,1239.
21,99.9119,2.4,4.5,617.427,559.441,559.441,617.623,559.47,559.47,99.9309,99.9634,99.9314,99.8985,647.626,107.732,0,109.909,53.9079,53.8983,1999.03,2004.9,1239.58,1239.5
22,99.8803,2.4,4.5,617.439,559.786,559.786,617.639,559.818,559.817,99.8243,99.8916,99.8256,99.8261,647.478,107.732,0,109.909,53.3451,53.3348,1769.79,1774.61,1239.32,123
23,99.2579,2.4,4.5,617.474,560.136,560.136,617.676,560.171,560.171,99.6782,99.7676,99.6802,99.7015,647.266,107.732,0,109.909,52.4309,52.4202,1511.46,1514.86,1238.69,123
24,98.1321,2.4,4.5,617.559,560.495,560.495,617.761,560.534,560.534,99.5417,99.6524,99.5437,99.5857,646.991,107.732,0,109.91,51.1908,51.1784,1259.32,1254.56,1238.09,1238
25,97.0197,2.4,4.5,617.69,560.847,560.847,617.891,560.895,560.895,99.4032,99.5318,99.4053,99.4649,646.654,107.732,0,109.91,49.7159,49.6991,976.667,971.777,1238.04,1238.
26,96.0738,2.4,4.5,617.831,561.214,561.214,618.03,561.27,561.27,99.2843,99.4238,99.2863,99.3572,646.275,107.732,0,109.91,47.9333,47.9031,660.544,654.734,1236.89,1236.9,
27,95.3078,2.4,4.5,617.971,561.706,561.706,618.168,561.77,561.77,99.1497,99.3005,99.1518,99.2335,645.87,107.732,0,109.91,45.7232,45.6752,264.081,254.555,1234.1,1234.12,
28,94.4749,2.4,4.5,618.11,562.291,562.291,618.305,562.358,562.357,98.9107,99.0682,98.9135,99.0008,645.442,107.733,0,109.91,43.2521,43.185,0,0,1229.61,1229.63,1232.73,12
```

Figure 26: Screenshot of Dataset Collected from GPWR

3.1.4 Data Compiling

In order for the data collected from the reactor simulator to be used to create a machine learning model, some data modifications needed to be performed. First, the data was in 39 separate datasets, to avoid issues with the constant moving, modifying, loading etc. of the data, these sets were combined into one complete dataset. This set consisted of 30,710 data points, each consisting of the 33 measured features and the features added for reactor core life and transient event. Also, to minimize confusion and ensure only the data was imported into the machine learning model, the feature labels and heading information was not included in the final dataset. These preparations were done using Microsoft Excel and the final dataset was saved as a CSV file. The 39 individual datasets will be maintained in the event any unexpected issues occur with the complete dataset.

3.1.5 Data Exploration and Modification Using Python

Once the data was compiled into a single dataset, Python was used to modify and explore the data. All code written for this part of the project was done using Python version 3.7.2, the most up-to-date version available at the time of performing the study. The scripts were written using Atom text editor and all code compiling was done using the Anaconda Python distribution. The complete CSV file was imported using Pandas. This converted the data from a CSV file into a Pandas DataFrame. No header was used in the importing of the data. The contents of the DataFrame was then verified using the .head(), .shape() and .describe() commands. The .head() command allows the user to view the contents of the first 5 rows of a dataset. This was done to ensure that all the features appear correctly in the DataFrame. The .describe() command provides the descriptive statistics of the data stored in the DataFrame. This includes the mean, standard deviation, data point count, as well as the minimum and maximum values of each feature. The summary statistics for the first and last 3 columns of the dataset are shown in Figure 27. This also allowed for verification that all the data points had imported into the DataFrame.

	0	1	2	...	19	20	21
count	30710.000000	30710.000000	30710.000000	...	30710.000000	30710.000000	30710.000000
mean	541.561225	521.515862	518.776119	...	2131.796682	590.801487	384.920847
std	105.355255	98.713384	107.142703	...	202.855889	136.808571	555.122586
min	183.631000	201.266000	98.079500	...	1700.000000	213.138000	-19.339300
25%	564.761000	559.310000	559.310000	...	2164.875000	648.124250	0.000000
50%	573.993000	562.153000	562.153000	...	2236.280000	652.786000	0.000025
75%	593.725750	564.770500	564.770500	...	2239.667500	653.004000	655.580000
max	622.519000	577.808000	577.808000	...	2314.580000	657.763000	1549.110000

Figure 27: Sample Descriptive Statistics from Initial Dataset

Analysis of the dataset's descriptive statistics showed that a number of the features collected were just percentages of actual values. For example, normalized neutron flux is simply a percentage of power generated. These features were deemed redundant for model training and dropped. Also, a couple of features were consistent throughout the dataset, such as the level of the narrow range of the RCS. These features were also dropped from the dataset. In total, 22 features were used in training these models. The final list of features used in training, as well as the maximum and minimum values for each feature are given in Table 7. The addition of the reactor core life introduced a qualitative feature into the dataset. Machine learning algorithms are only able to use quantitative data to produce a model. In order to properly account for the reactor lifetime, it was necessary to convert the qualitative data into quantitative data; this was done using dummy variables. Dummy variables are typically used to represent qualitative data in a 0, 1 scale. In this case, since there are three different types of qualitative data (BOL, MOL and EOL), three dummy variables and two extra factors were needed. It was possible to convert this data using the Pandas function .get_dummies. This function was used to create a dummy variable DataFrame using the reactor core life column of the dataset. The dummy variable DataFrame consists of 2 columns. BOL data points were converted to 0,0, EOL data points were converted to 1,0 and MOL were converted to 0,1. The dummy variable DataFrame was then added to the end of the dataset using Pandas' .concat function, which is used to merge two or more DataFrames. Finally, the original reactor core lifetime column was dropped from the DataFrame. To ensure that the process had been done correctly, the new DataFrame was explored once again. The next step in preparing

Table 7: Final Features used in Initial Dataset

Feature	Minimum Value	Max Value
Hot Leg 1 Temperature (F)	183.631	622.519
Hot Leg 2 Temperature (F)	201.266	577.808
Cold Leg 1A Temperature (F)	98.079	577.808
Cold Leg 1B Temperature (F)	220.353	622.652
Cold Leg 2A Temperature (F)	122.800	577.435
Cold Leg 2B Temperature (F)	101.278	577.435
Pressurizer Surge Line Temperature (F)	181.631	651.354
PORV Discharge Pressurizer Temperature (F)	107.736	117.229
Containment Pressure (PSI)	0.0	108.244
Containment Temperature (F)	89.4057	246.052
FW Flow to SG-1 (LB/S)	0.0	2508.38
FW Flow to SG-2 (LB/S)	0.0	2582.82
MS Flow from SG-1 Line-1A (LB/S)	0.0	1244.16
MS Flow from SG-1 Line-1B (LB/S)	0.0	1244.21
MS Flow from SG-1 Line-2A (LB/S)	0.0	1293.26
MS Flow from SG-1 Line-2B (LB/S)	0.0	1295.33
SG-1 Pressure (PSI)	117.753	1265.71
SG-2 Pressure (PSI)	183.387	1268.57
Average Temperature (F)	179.241	598.453
Pressurizer Pressure (PSI)	1700	2314.58
Pressurizer Steam Temperature (F)	213.138	657.763
Generator Power (MW)	-19.3393 (Consuming Power)	1549.11
Reactor Core Life	N/A	N/A

the data was to prepare the target data of the dataset. As mentioned earlier, each data point was given a label of the transient event that occurred when the data was collected. This column was also a qualitative feature. Unlike the reactor core lifetime, there was no need to use dummy variables when modifying this dataset. Instead, each transient was designated a number: the feed water pump trip was assigned 1, the LOCA-LOOP 2, the steam generator valve closure was assigned 3 and the rapid power change was assigned a 4. Normal operations were assigned 0. Using Pandas' .map function it was possible to change all the qualitative data to the assigned numerical value. The dataset was once again explored to ensure that the process had been implemented correctly.

3.1.6 Data Splitting

The final step in preparing the reactor simulator dataset was to split the dataset into a training set and a testing set. In supervised machine learning, data should be split in order to validate the results. Validation allows for a measurement on the quality of the model's results. In the case of this project, validation is critical. As mentioned, regulatory agencies, such as the NRC, have strict requirements in proving that any system or component within a reactor will behave as it is intended, especially if it will be relied upon in abnormal events. An important aspect of validation is that the data used in the testing must be completely independent of the data used in creating the model. Failure to ensure this could result in biased models that do not learn the actual case of the testing data.

It is important to balance how much of the data is split between the two sets. If too little data is put into the training dataset, the algorithm will not be able to learn the differences between the data points. This will result in less accurate models, which will be less effective in performing the task intended for the model. It is also necessary to have enough testing data. If the algorithm lacks sufficient testing data it will be difficult to verify that the model created by the supervised learning algorithm is reliable. Finally, as is the case in most statistical procedures, it is important that the data splitting be random to avoid any biases and to provide a good sample for both, the testing and training sets.

The data splitting for this project was done using scikit-learn's tools. This is done using the test_train_split function. This function uses Bernoulli sampling in order to create testing and training sets that are pseudo-random. The pseudo-random nature of the splitting allows for the process to be repeated over and over again with no changes to the outcome while maintaining the randomness of the selection. The function requires that features and target data be provided as well, as the desired split between testing and training data. Also, the user may specify the seed of the random number generator, if desired. The output will be four different NumPy arrays, two arrays for the feature and

```
#Splitting the Dataset into Training and Testing Set 50 50 Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.5, test_size=0.5)
```

Figure 28: Test Train Split Code for Initial Dataset

Anaconda Prompt

	0	1	2	3	4	5	...	18	19	20	21	EOL	MOL
6056	317.653	311.901	312.747	318.659	313.388	313.390	...	315.506	1700.00	312.438	0.002501	0	0
16576	574.213	573.634	573.634	574.153	573.501	573.501	...	573.875	2244.17	653.292	0.000000	1	0
19674	610.316	557.014	557.014	610.530	557.040	557.040	...	583.725	2241.72	653.137	1280.290000	0	1
22940	617.447	559.334	559.334	617.646	559.362	559.362	...	588.447	2237.20	652.846	1408.760000	0	1
30399	564.379	563.654	563.654	564.378	563.654	563.654	...	564.016	2234.63	652.678	0.000000	0	1
13739	587.883	574.284	574.284	587.700	574.365	574.365	...	581.058	2220.74	651.787	-17.126200	0	1
14877	576.562	574.370	574.370	576.545	574.084	574.084	...	575.390	2248.07	653.544	0.000000	1	0
21232	617.605	559.423	559.423	617.638	559.427	559.427	...	588.523	2236.25	652.786	1409.050000	1	0
27405	564.761	564.045	564.045	564.762	564.045	564.045	...	564.403	2236.89	652.824	0.000000	0	0
8770	458.462	404.148	401.678	467.558	418.389	418.285	...	436.818	1700.00	333.142	0.029798	0	1
24697	591.165	560.611	560.611	591.411	560.592	560.592	...	575.945	2250.43	653.695	648.321000	0	1
25527	591.490	561.084	561.084	591.755	561.079	561.079	...	576.352	2239.82	653.013	650.836000	0	1
18130	611.693	561.963	561.963	611.715	561.964	561.964	...	586.833	2215.38	651.436	1166.030000	1	0
7327	312.048	306.300	307.045	313.138	308.057	308.059	...	309.979	1700.00	304.131	0.000431	0	1
186	568.802	563.859	563.859	568.726	563.805	563.805	...	566.298	2063.04	641.254	0.003355	1	0
8070	594.125	563.299	563.299	594.180	563.297	563.297	...	578.725	2243.66	653.261	656.749000	1	0
27286	564.761	564.044	564.044	564.761	564.044	564.045	...	564.403	2236.92	652.826	0.000000	0	0
19327	606.970	561.434	561.434	607.165	561.448	561.448	...	584.254	2217.78	651.592	1043.970000	0	1
19114	613.023	561.916	561.916	613.194	561.931	561.931	...	587.516	2223.59	651.968	1206.200000	0	1
587	567.936	565.457	565.457	567.921	565.395	565.395	...	566.677	2209.13	651.026	0.000000	1	0
5511	564.377	563.652	563.652	564.376	563.652	563.652	...	564.014	2234.85	652.693	0.000000	0	1
25155	591.547	561.092	561.092	591.801	561.087	561.087	...	576.382	2243.76	653.266	651.070000	0	1
20169	617.427	559.311	559.311	617.656	559.343	559.343	...	588.434	2238.16	652.908	1409.020000	0	0
25271	591.541	561.099	561.099	591.798	561.095	561.095	...	576.383	2242.51	653.186	650.970000	0	1
20273	617.425	559.309	559.309	617.654	559.341	559.341	...	588.432	2238.37	652.922	1408.980000	0	0
17892	617.103	560.382	560.382	617.135	560.386	560.386	...	588.752	2239.18	652.974	1353.600000	1	0
11714	576.514	574.310	574.310	576.497	574.024	574.024	...	575.336	2248.02	653.541	0.000000	0	0
8903	339.128	317.950	318.078	342.911	319.285	319.282	...	329.834	1700.00	309.511	0.000030	0	1
10328	254.102	265.938	245.809	272.338	223.598	194.993	...	247.902	1700.00	242.479	0.000000	1	0

Figure 29: Sample from X Train Dataset

target training data and two arrays will be for the feature and target testing data. These were labeled as X_test and X_train for the feature data and Y_test and Y_train for the target data. For this model, the target data will be the numerically-labeled transient types and the feature data will be the 22 features collected from the reactor simulator. Half of the data collected will be used for training and the other half will be used for testing. The default random number seed for this function will be used for all data splitting on this phase of the project. The Python code used is shown in Figure 28. Figure 29 shows the portion for the X_train array.

3.2 Results

The TPOT Classifier specified in the previous chapter was then used to train the six machine learning models. Each model was scored with using the four validation measure-

ments: accuracy, precision, recall and F1 score. The results from each of these models will be presented in this section.

3.2.1 K-Nearest Neighbors

The entire process of building and evaluating the k-nearest neighbors model in TPOT took approximately 1 hour and 30 minutes. The accuracy of this model was 98.35%, the precision was 98.02%, recall was calculated to be 98.01%, and the F1 score was also calculated to be 98.01%. Table 8 shows the individual accuracies for each transient from this model.

Table 8: K-Nearest Neighbors Initial Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	96.6%
LOCA + LOOP	97.57%
Valve Closure	98.01%
Rapid Power Change	97.86%

The k-nearest neighbors method was able to correctly identify 15,103 instances of the 15,355 samples tested during the validation process. Of the 252 misclassified instances, the largest amounts of misclassifications were from the feed water pump trip transient. 172 of the 252 misclassifications, 60% of total errors, were from this transient. Of those 172 errors, 95 were false positives and 77 were false negatives. The model's biggest issue was distinguishing the feed water pump trip transient from the valve closure transient: a total of 76 instances, 30% of the total misclassifications were between these two transients. The k-nearest neighbors model was able to perfectly distinguish normal operation instances from transient instances, as there were no type I or type II errors for the normal operation transient. Figure 30 shows the confusion matrix for the k-nearest neighbor model. The code was designed to export the instances where misclassifications occurred. Initial analysis of these instances showed no true pattern or bias of when in the

transient the misclassifications occurred.

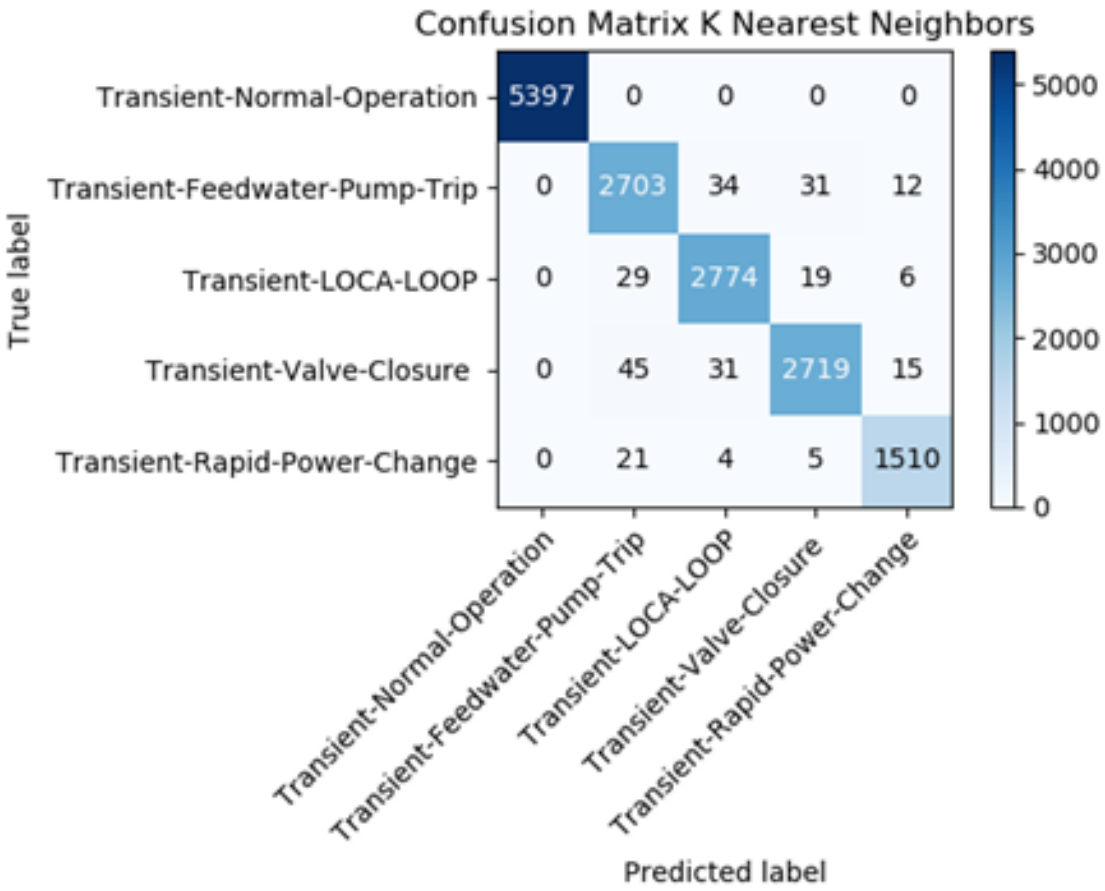


Figure 30: Confusion Matrix for Initial K-Nearest Neighbors Model

3.2.2 Bernoulli Naïve Bayes Results

Using TPOT, the Bernoulli naïve Bayes model took approximately 1 hour to build and validate. The accuracy of this model was 97.45%, the precision was calculated to be 97.18%, the recall of the model was 96.73 %, and the F1 score was 96.87%. Table 9 shows the accuracies of the individual transients from this model.

The Bernoulli naïve Bayes model correctly identified 14,964 instances of the 15,355 tested. Of the 391 incorrect classifications, 258 of them or 66% of total misclassifications, were from the feed water pump trip transient. Of these, 191 were false positives for the valve closure transient. The model also scored 33 false positive classifications for the rapid power change transient. Under this configuration, the model was able to cor-

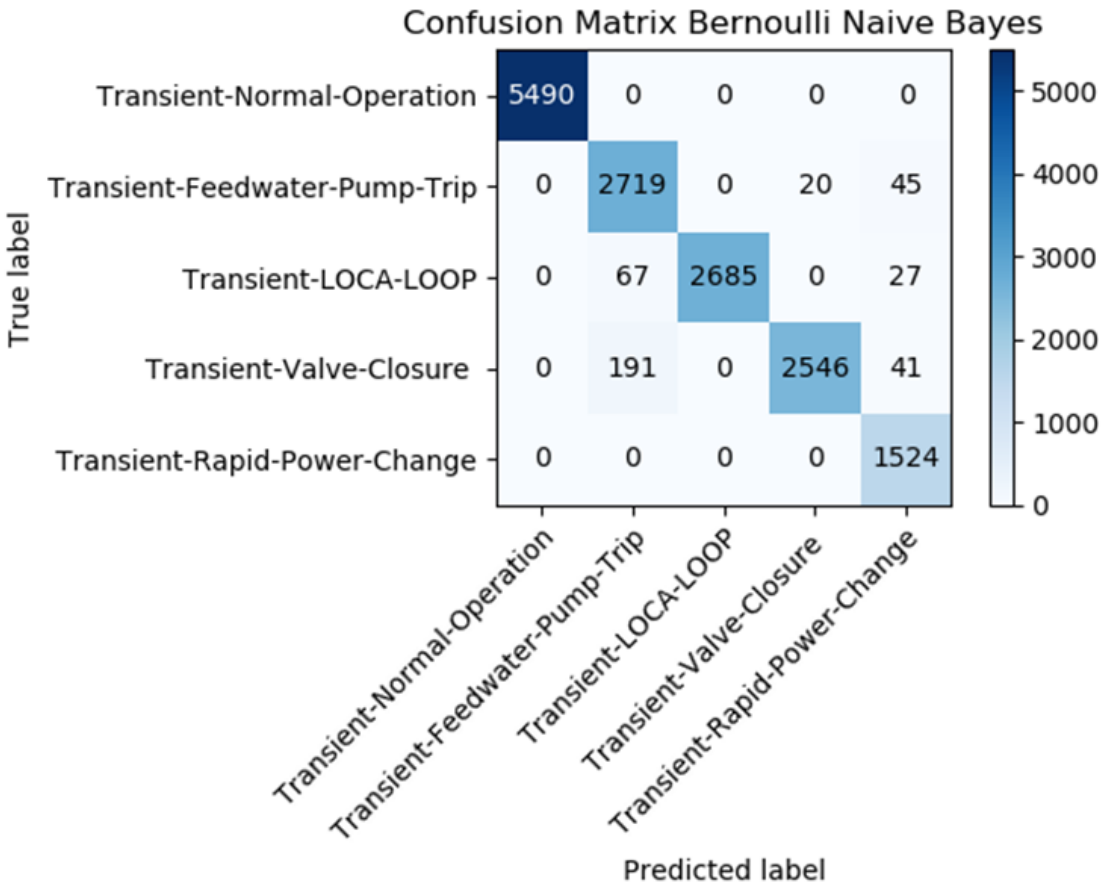


Figure 31: Confusion Matrix for Initial Bernoulli Naïve Model

rectly distinguish between a transient and non-transient event with no Type I or Type II errors for the normal operations event. The Bernoulli naïve Bayes model had no Type I errors for the LOCA-LOOP transient, nor were there any Type II errors for the rapid power change transient. The confusion matrix for the Bernoulli naïve Bayes model is shown in Figure 31.

Table 9: Bernoulli Naïve Bayes Initial Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	90.05%
LOCA + LOOP	100%
Valve Closure	96.1%
Rapid Power Change	93.72%

3.2.3 Gaussian Naïve Bayes Results

Similar to the Bernoulli naïve Bayes model, the Gaussian naïve Bayes model took approximately 1 hour to build and test. The accuracy of this model was found to be 97.45%. The precision was scored at 97.2%, the recall was calculated at 96.83%, and the F1 score was 96.96%. Table 10 shows the model's accuracy for the individual transients. This model was able to correctly identify 14,833 of the 15,355 samples tested. The Gaussian naïve Bayes model performed perfectly in identifying the non-transients and transients, as there were no false positives or negatives for the normal operation event. There were also no false positives for the LOCA LOOP transient and no false negative for the rapid power change transient. The model struggled the most with correctly classifying the feed water pump trip transient. Of the 522 misclassified transients, 337 of them or nearly 65% of all the model's total errors were from this transient. Of those, 302 were misclassifications between the feed water pump transient and the valve closure transient. The confusion matrix for the Gaussian naïve Bayes model is shown in Figure 32.

Table 10: Gaussian Naïve Bayes Initial Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	88.77%
LOCA + LOOP	100%
Valve Closure	96.59%
Rapid Power Change	94.0%

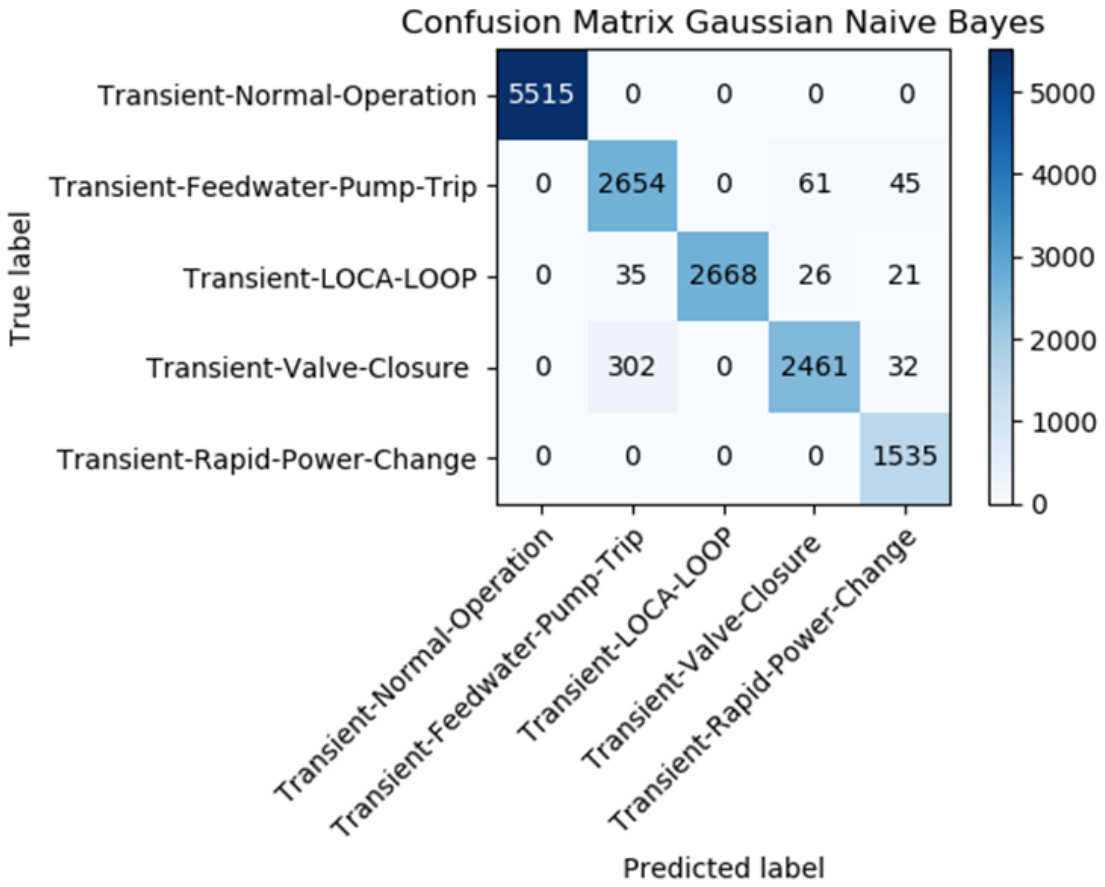


Figure 32: Confusion Matrix for Initial Gaussian Naïve Model

3.2.4 Multinomial Naïve Bayes Results

The multinomial naïve Bayes model took approximately 1 hour to be built and tested using TPOT. The accuracy of the multinomial naïve Bayes model was 96.71%. The precision of this model was calculated to be 96.38%, the recall was 95.41%, and the F1 score was calculated to be 96.10%. Table 11 shows the accuracies of the individual transients from this model.

The multinomial naïve Bayes model was able to correctly classify 14,833 of the reactor transient instances tested. Similar to the other naïve Bayes models, the multinomial method was able to perfectly distinguish between transient events and non-transient events, as there were no Type I or Type II errors for normal operations. The rapid power

Table 11: Multinominal Naïve Bayes Initial Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	90.05%
LOCA + LOOP	100%
Valve Closure	96.1%
Rapid Power Change	93.72%

transient also had no false negative results and the LOCA LOOP transient had no false positives. Also, the model struggled most with the feed water pump trip transient with 296 misclassifications occurring with this transient. The confusion matrix for this model is shown in Figure 33.

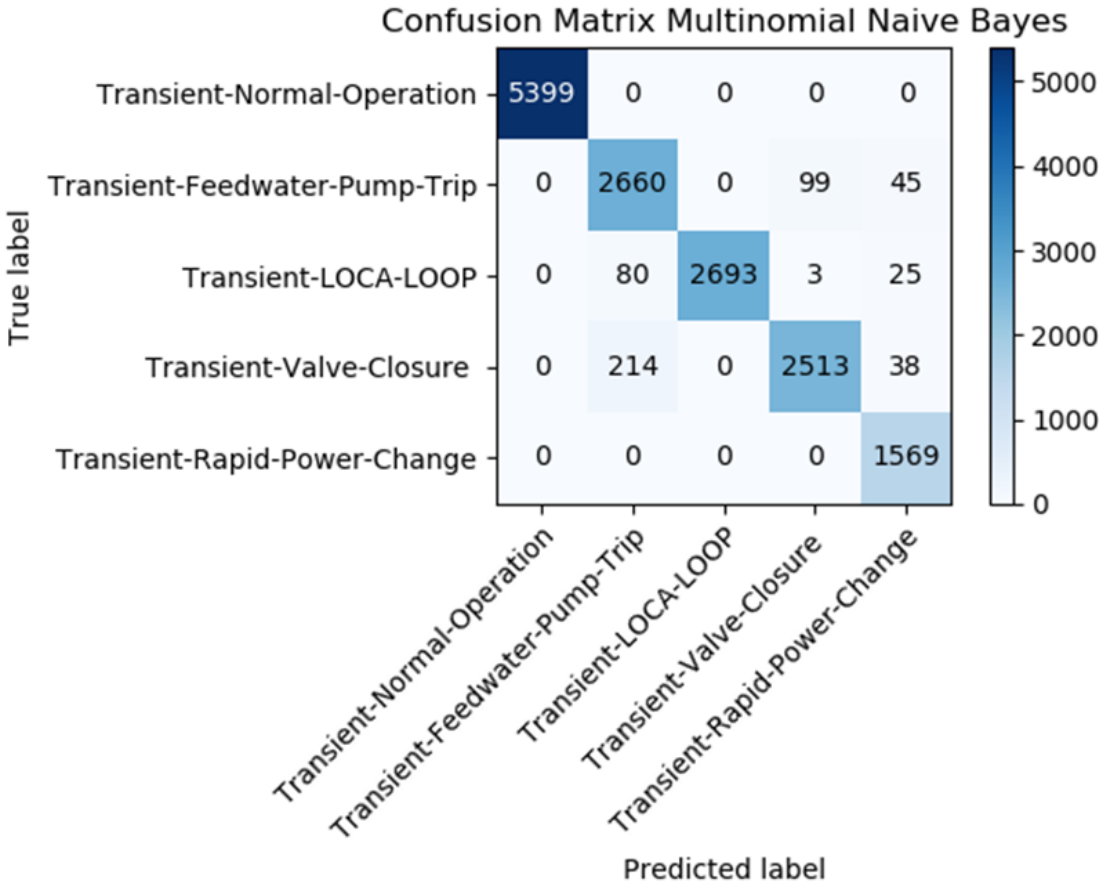


Figure 33: Confusion Matrix for Initial Multinominal Naïve Bayes Model

3.2.5 Logistic Regression

The logistic regression model took approximately 48 hours to run. This was the most computationally expensive of all the models evaluated. The accuracy of the logistic regression model was found to be 98.55%. The precision was calculated to be 98.41%, recall was 98.04% and the F1 score was found to be 98.21%. Table 12 shows the individual accuracies for the reactor transients for this. This model correctly identified 15,133 transient instances of the 15,355 samples tested. The logistic regression model perfectly classified transient and non-transient events; there were no false positives or negatives from the normal operation event. The model also had no false positives for the LOCA LOOP and there were no false negatives for the rapid power change. The model scored well on all the transients with accuracies above 95% across all 5 events. The model had the highest number of misclassifications with the feed water pump, though the rapid power change had a lower accuracy. The largest number of errors, 52, came from false positives of the feed water pump transient from the valve closure transient. Figure 38 shows the confusion matrix for this model. No easily identified groups were found when looking at the misclassified instances, similar to some of the other models the misclassifications appear spread out. Figure 34 shows a graph of the misclassifications.

Table 12: Logistic Regression Initial Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	96.87%
LOCA + LOOP	100%
Valve Closure	97.62%
Rapid Power Change	95.71%

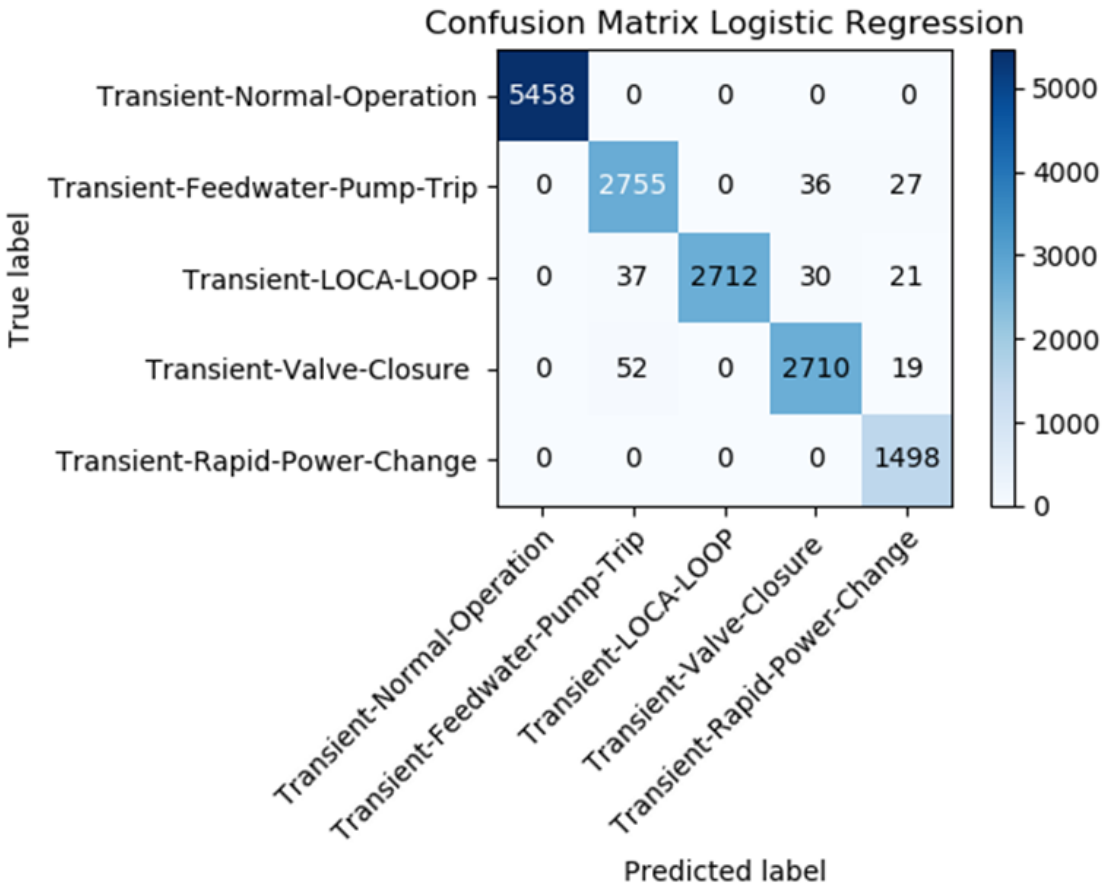


Figure 34: Confusion Matrix for Initial Logistic Regression Model

3.2.6 Decision Tree

The decision tree model took approximately 3 hours to build and validate. The accuracy of this model was 98.6%, precision was calculated at 98.46%, recall was found to be 98.1% and the F1 score was 98.27%. Table 13 shows the individual accuracies of the transient events for the decision tree model. The decision tree model was able to classify correctly 15,140 of the 15,355 transient instances tested. The model was able to perfectly classify all of the normal operation instances and there were no false positive or false negative errors from that event. The model was able to classify the LOCA LOOP transient with no false positives and the rapid power change transient had no false negatives. As with the other models, the decision tree model's biggest issues were from the feed water pump transient: 35% of the errors and 76 instances were from this transient. 41 of those were false positives with the valve closure transient. The confusion matrix for this model is shown in Figure 35 . Looking at misclassified instances, no obvious grouping appeared. With the exception of the Bernoulli naïve Bayes model, it appears that the misclassifications experienced were typically spread out rather than grouped together.

Table 13: Initial Decision Tree Model Individual Accurcies

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	97.33%
LOCA + LOOP	100%
Valve Closure	97.57%
Rapid Power Change	95.62%

3.3 Discussion

3.3.1 Overall Model Performance

The results from the machine learning models show very positive results. All of the models had validation scores in the mid-90's. Under the configurations selected for the TPOT

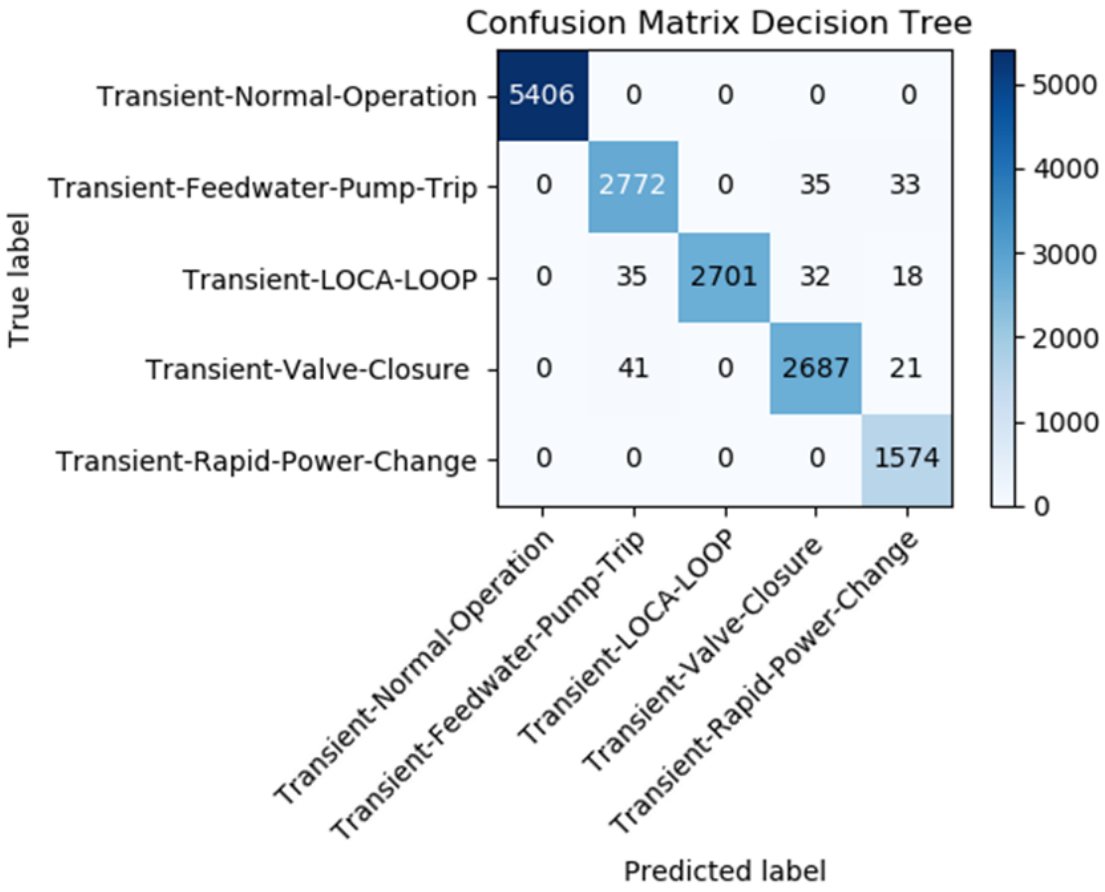


Figure 35: Confusion Matrix for Initial Decision Tree Model

dictionary, all of the models were able to perfectly tell the difference between normal operations and transient events. It should be noted that the dataset did contain more of this type of data, but this should not be an issue, as with real nuclear reactors the amount of data, as well as the quality, will almost certainly be higher for a real reactor under normal operations. With the exception of the k-nearest neighbors model, the models were able to correctly classify the LOCA LOOP transient with perfect accuracy, though there were false negative classifications across all these models in the study. All of the models had the most difficulty distinguishing between the feed water pump trip and the valve closure transients, as this transient had the lowest individual accuracy of the five events across all the models. Also, a large percentage of the total errors from these models came from false positives between these two transients. The models appear to have a tendency towards having more Type II error over Type 1, as the precision of all the models is higher than the recall. Since Type II error can result in a more dangerous scenario with a nuclear reactor, it is important that the recall always be considered when making determinations. Table 14 and Table 15 summarize all the validation results from this part of the project.

Table 14: Summary of Machine Learning Model Results: Overall Validation Measurements

	Accuracy	Precision	Recall	F1 Score	Time
K-Nearest Neighbors	98.35%	98.02%	98.01%	98.01%	1.5 hrs
Bernoulli naïve Bayes	97.45%	97.18%	96.73%	96.87%	1 hr
Gaussian naïve Bayes	97.45%	97.2%	96.83%	96.96%	1 hr
Multinomial naïve Bayes	96.71%	96.38%	95.41%	96.10%	1
Logistic Regression	98.55%	98.41%	98.04%	98.21%	48 hrs
Decision Trees	98.6%	98.46%	98.1%	98.27%	3 hrs

3.3.2 Best Performing Models

In terms of performance, the decision tree, k-nearest neighbors and logistic regression models were better than the naïve Bayes models, having validation measurements all above 98%. The decision tree model performed the best. Interestingly, the decision tree

Table 15: Summary of Machine Learning Model Results: Individual Transient Accuracies

	Normal Operation	Feedwater Pump Trip	LOCA + LOOP	Valve Closure	Rapid Power Change
K-Nearest Neighbors	100%	96.6%	97.57%	98.01%	97.86%
Bernoulli naïve Bayes	100%	90.05%	100%	96.1%	93.72%
Gaussian naïve Bayes	100%	88.77%	100%	96.59%	94.0%
Multinomial naïve Bayes	100%	90.05%	100%	96.1%	93.72%
Logistic Regression	100%	96.87%	100%	97.62%	95.71%
Decision Trees	100%	97.33%	100%	97.57%	95.62%

model slightly outperformed the logistic regression model. This is important as the logistic regression model is more computationally expensive, requiring 48 hours to compute on its initial run, while the decision tree model only took 3 hours. In this case, it is likely the data was not overly complex, so the decision tree model was able to better fit the data without needing to perform the large amount of feature analysis required with the logistic regression model. This is encouraging, as time is a major consideration when selecting a model that will be used in real-time. Using more sophisticated equipment, it is possible that a decision tree or k-nearest neighbors model could provide valuable information to reactor operators in a matter of minutes, if a model needs to be trained quickly. A TPOT model was constructed early on in the experiment without defining a specific machine learning model to be built. The optimal model from this run was a decision tree, with an accuracy of 98.57%. This verifies the results from the individual tests, where the decision tree was the most accurate model.

3.3.3 Models with Potential Issues

The naïve Bayes models, while having high validation measurements, did not perform as well as the other three models. The multinomial model had the lowest accuracy of the six models. A likely cause of this is that the data better fit the Gaussian and Bernoulli distributions than the multinomial. The accuracies with the feed water pump transient were the lowest with the Gaussian model, only scoring 88% accuracy for that transient.

The most likely cause of this is that the probabilities calculated by the naïve Bayes models favored the feed water transient over the valve closure, resulting in the false positive classifications. The overall accuracies of the Gaussian and Bernoulli models were identical. This is likely coincidental, as both models have different accuracies for three of the individual transients.

3.3.4 Final Thoughts

Due to the high validation measurements, it does appear from this analysis that there is promise in the area of applying machine learning to reactor safety. Applying trained machine learning models to reactor safety could lead to faster transient diagnoses, accident mitigation, and help keep the general public better informed of issues at a nuclear power plant. Areas that will be further explored include introducing more transients to see how the models perform and which models perform better with the more complex data. Also, further exploration into the errors within these models to look for more patterns and factors behind the existing errors, will be done. Finally, other machine learning models, such as those using ensemble methods will be trained to see if they can perform better.

4 Expanded Dataset & Optimal Model Analysis

The first phase of this project was presented at the American Nuclear Society's Annual Conference in the Fall of 2019. The final results were also published in the journal Nuclear Technology in the Spring of 2021. This was done in order to gather feedback on possible improvements that could be explored for the next phase of the project. One of the comments on the initial phase of this research effort was the limited number of transient events the models were trained on. The primary goal of this phase of the project is to test the ability of TPOT to train models with a larger number of transient events. In addition to the original six methods used in the previous phase of the project, this study will look to train a model using Random Forest Classification.

Another area of interest for this phase of the project is to address some of the possible concerns that the implementation of a machine learning based diagnostic system may encounter. As mentioned earlier, systems used in the area of nuclear safety must undergo a great deal of scrutiny by regulatory authorities such as the NRC. This phase of the project will take a more in depth look at where misclassifications occurred with the models. Another area of concern with a safety related system is what to do in the event certain data is unavailable when a transient occurs. Finally, as many machine learning models make use of random numbers in some form, the variation of models will be examined. The goals of this phase of the project are summarized below.

Goals of Second Phase of the Effort

1. Use of an expanded dataset to determine if models can be relied upon with a great number of transient events.
2. Train a model using Random Forest Classification.
3. Examine where in the simulations, classifications occurred to determine if major patterns or concerns exist.
4. Train models with key features missing to determine if models can be trained and

relied on if features are missing during a transient.

5. Examine the effect changes in random states have on the validation results of high performing models.

4.1 Ensemble Learning

In the area of machine learning, Ensemble Learning has quickly become on of the most powerful techniques in producing high scoring models. The idea of Ensemble Learning is to collect results from several different models and aggregate the collective results. This has the ability to take models that perform relatively poorly and create a better performing model. Ensemble approaches have become popular in recent times due to this capability.

One example of a well known Ensemble learning model was the winning submission in a competition held by Netflix in the late 2000s. Netflix in an effort to improve its movie recommender algorithm, held a three year competition to see what improvements could be made. The winning model is known as BellKor solution, named after its creators[74]. This model was an Ensemble method blending several types of regression models, most prominently k-nearest neighbors. The results produced a RMSE of 0.8712, a near 10% improvement from Netflix's algorithm.

There are a number of Ensemble Learning techniques available in scikit-learn. A commonly used one is the Voting Classifier, which selects the result that receives the majority of votes from the different models. Another approach is called Bagging, which makes use of the same classifier, but uses different subsets of the training data, similar to cross validation. Another ensemble method that can be used to improve models is gradient boosting. Typically used in regression, gradient boosting uses weak learners and the errors from those learners to compute a residual. The model is then trained on the residual and the model then tries to predict those residuals. It should be noted that boosting is a greedy algorithm and can produce overfit. models[75].

4.1.1 Random Forest

This project will make use of the Random Forest technique. Random Forest is an averaging Ensemble method that can be used in either regression or classification. This method constructs several different decision trees rather than just a single tree. The results from each of the different trees are averaged and a final result is determined. In classification, the result is determined by which result received the most votes from the individual tress in the forest. Regression uses the average of output to determine the final result. The use of random forest can reduce overfitting and improve accuracy. Random Forest makes use of many of the same hyperparameters that a single decision tree uses[76]. This includes max depth, criterion and max features. As was the case in the previous study, Gini Impurity will be used. TPOT has full compatibility with scikit-learn's Random Forest Classifier and Random Forest Regressor.

4.2 Methodology

4.2.1 Expanding The Dataset

In the original study, data was collected on a reactor operating normally and experiencing four individual transient events. These transients were: a simultaneous trip of all reactor feed water pumps, simultaneous closure of reactor steam isolation values, large break LOCA coupled with a loss of offsite power, rapid power change from 100% to 75% and back to 100%. Data was recollected for these transients using the GPWR simulator. In addition, data was collected on seven additional transients. These were: trip of single reactor coolant pump, simultaneous trip of all reactor coolant pumps, a rapid depressurization, a max steam line rupture, a turbine trip without a SCRAM, a rejection of electrical load and an accidental manual reactor trip. Table 16 lists all the transient events simulated for this study. A brief description of each of the new transient events is given in the next subsection.

Table 16: Transient Events Simulated from GPWR .

Normal Operations	Simultaneous trip of all reactor feed water pumps
Large Break LOCA coupled with a loss of offsite power	Simultaneous closure of reactor steam isolation values
Rapid Power change from 100% to 75% and back to 100%	Trip of Single reactor coolant pump
Simultaneous trip of all reactor coolant pumps	Rapid Depressurization
Max Steam Line Rupture	Turbine trip without a SCRAM
Rejection of electrical load	Accidental manual reactor trip

In order for the dataset to have a more complete picture of reactor behavior, it was also necessary to collect data using different initial conditions. In the original study, nine initial conditions were used. These consisted of different combinations of reactor power level and reactor core life. In this study, six additional initial conditions were simulated. A BOL core at 25% power at xenon equilibrium, a MOL core at 5% power in a startup configuration, a MOL core at 15% power in a startup configuration, as well as a sub-critical core at 1% with all three reactor core lives. Table 17 lists the 15 initial conditions used in the data collection. Most simulations with the GPWR ran for 600 seconds. The rapid power change transient however, required more than 1000 seconds to complete the simulation and due to the nature of the transient, only three simulations were performed. Since it is likely that any actual implementation would have access to more data on normally operating reactors, the normal operations simulation was run for 1,200 seconds. This does create an unbalanced dataset, but is realistic for this type of application.

Table 17: Initial Conditions Used for GPWR Simulation .

Simulation #	**Core Life**	**Power Level**
1	BOL	100% Power
2	MOL	100% Power
3	EOL	100% Power
4	BOL	50% Power
5	MOL	50% Power
6	EOL	50% Power
7	BOL	1% Power, (Critical, Startup Configuration)

Table 17: Initial Conditions Used for GPWR Simulation .

Simulation #	Core Life	Power Level
8	MOL	1% Power, (Critical, Startup Configuration)
9	EOL	1% Power, (Critical, Startup Configuration)
10	BOL	1% Power, (Sub-critical, Shutdown Configuration)
11	MOL	1% Power, (Sub-critical, Shutdown Configuration)
12	EOL	1% Power, (Sub-critical, Shutdown Configuration)
13	BOL	25% Power, (Critical at Xe Equilibrium)
14	MOL	5% Power, (Critical at Startup Configuration)
15	MOL	15% Power, (Critical at Startup Configuration)

4.2.2 New Transient Events

As mentioned, this phase of the project made use of seven new simulated transient events. This section will briefly describe each of these new events. The first of these is a manual reactor trip. In this simulation, once the trip is initiated the neutron flux is expected to drop from whichever level it was at to a delayed neutron state for a sub-critical system. Parameters of interest include the pressurizer level and power, as well as average temperature, reactor power and hot leg temperature from any loop. The next added transient for this experiment was the the trip of a single reactor coolant pump. In this transient, it is expected that the temperature of the moderator and coolant, water in both cases, will begin to increase, to provide a negative net reactivity. The flux of the reactor should decrease to a sub-critical state if the system is not already in this state. In addition, simulations were run where all reactor coolant pumps trip. As was the case with a single trip, moderator temperature is expected to increase and the reactor should SCRAM to reach a sub-critical state. Parameters of interest for these transients include steam and feedwater flow, cold/hot leg temperature and neutron flux. The fourth new reactor transient simulated for this experiment was the tripping of the main turbine without a reactor SCRAM. In this transient, reactor power will still decrease due to the steam dump and the automated rod controller. Power is expected to drop to 30% if the system is not

already at or below this level. Parameters of major interest include neutron flux, average temperature and the steam generator pressure and level.

The next new transient used for this experiment was a maximum steam line rupture. In this transient, neutron flux is expected to decrease to sub-critical levels if not already in this state. Containment temperature will increase rapidly at first, but should decrease as the simulation continues. Pressure within the system should rapidly decrease to that of the containment. Other parameters of interest include the pressure of the narrow range pressurizer and level of the pressurizer.

The sixth new transient event introduced in this study is a slow depressurization of the reactor's primary system. During this transient neutron flux will decrease slowly for a short time until the reactor SCRAMS. Pressurizer pressure should decrease to the saturation level. The level of the pressurizer is expected to reach a solid state within five minutes of the simulation. Other areas of interest include the loop levels, the surge line temperature and temperature of the hot legs. The final new transient simulated for this effort was a maximum design load reject. At the start of this simulation both main breakers of the reactor are opened. The reactor is expected to SCRAM if it is not in a sub-critical state. The average temperature of the system will increase at first, but decrease rapidly due to steam dumping. Pressure in the system will increase, but the increase should be limited to heat removal activities. Areas of interest include the neutron flux, pressurizer level, temperature and pressure, as well as steam flow.

4.2.3 Data Exploration

In the previous study data on 33 reactor features was collected during each second of the simulation. These features were chosen because they are features that a reactor operator would generally have quick and easy access to. Since the models trained in that study were able to produce reliable measurements, the same features were collected during the new simulations. Table 4 lists all of the features that were collected for this part of the

study. The data collection resulted in 168 individual datasets, 15 for each event, except the rapid power change. These were combined into a single dataset to make exploration and model training simpler. This set consisted of more than 110,000 data points. This is more than triple than the 30,000 used in the previous study.

As is the case with most machine learning projects, the first step to to analyze the dataset prior to performing any model training. This was done using basic Pandas functions. This analysis ensured that there were no missing values for any of the features in the dataset and allowed for the examination of the dataset's descriptive statistics. The analysis also ensured that all the features were in the correct data type and helped in determining if any major outliers were present in the set.Once this analysis was completed, the next step was to determine which feature the model would be trained on. As was the case with the initial dataset, the normalized values, such as power and pressure, were dropped from the dataset as they are redundant for model training.

The next step in the data exploration was to determine if any of the features in the dataset were consistent, where the feature value is unchanged through the dataset. In the previous study, two features were found to be consistent, however, this was not the case with the expanded dataset. As such, these features remained in the set for training. The complete list of 25 features used in model training is listed in Table 18. Also included in this table are the ranges for all the features collected.

Table 18: Features used in in Expanded Model Training

Feature	Minimum Value	Max Value
RCS LVL LOOP 1 WR	0	2.4
RCS LVL LOOP 1 NR	2.4	4.5
Hot Leg 1 Temperature (F)	4.5	623.947
Hot Leg 2 Temperature (F)	199.179	624.07
Cold Leg 1A Temperature (F)	140.984	617.622
Cold Leg 1B Temperature (F)	109.407	577.809
Cold Leg 2A Temperature (F)	111.09	617.659
Cold Leg 2B Temperature (F)	99.2419	577.4340
Pressurizer Surge Line Temperature (F)	99.4619	651.343
PORV Discharge Pressurizer Temperature (F)	107.732	648.149
Containment Pressure (PSI)	0	108.297
Containment Temperature (F)	86.2528	246.051
FW Flow to SG-1 (LB/S)	0	2623.08
FW Flow to SG-2 (LB/S)	0	2583.99
MS Flow from SG-1 Line-1A (LB/S)	0	2000
MS Flow from SG-1 Line-1B (LB/S)	0	1244.73
MS Flow from SG-1 Line-2A (LB/S)	0	1645.35
MS Flow from SG-1 Line-2B (LB/S)	0	1645.78
SG-1 Pressure (PSI)	26.9171	1273.78
SG-2 Pressure (PSI)	40.8375	1275.66
Average Temperature (F)	175.722	598.453
Pressurizer Pressure (PSI)	1700	2318.73
Pressurizer Steam Temperature (F)	153.072	658.021
Generator Power (MW)	-19.3392 (Consuming Power)	1549.1
Reactor Core Life	N/A	N/A

As was the case in the previous chapter, it was necessary to convert the reactor core life feature to a numerical dummy variable, using scikit-learn's dummy variable function. Next, the transient data needed to be labeled numerically. Each transient was given a number between 1 and 11. Normal operating data points were assigned the label 0. Once these changes were made, the dataset was examined again to ensure the changes were implemented properly. The final step prior to model training was to split the data into testing and training sets. This was done once again using scikit-learn's test_train_split function. In machine learning, there is no standard on how much data to set aside for

testing and how much to use for data. To ensure the model will be changed adequately and help address possible overfit issues, the data was again randomly divided in two. At this point, the time stamp feature was dropped from both, the training and testing, as the data is irrelevant for the model training. It should be noted that this information is necessary to determine where in the simulation misclassifications occurred. As such, this feature was set aside to be used after training. The training and testing dataset sizes were checked to ensure the splitting was done properly.

4.2.4 Training New Models Using TPOT

The approach for training the machine learning models on the expanded dataset was similar to that used in the previous study. For this study, the same feature selection and data reprocessing techniques used in the previous study were used again. The complete list of feature prepossessing techniques used in the TPOT dictionary is given in Table 2. These functions are commonly used in machine learning efforts and provided high results in the previous study. In the previous study, six different types of models were trained. These were: decision tree classification, k-nearest neighbors, logistic regression and naive Bayes classification, using three different distributions: Gaussian, Bernoulli and multi-nominal. Since all six of these models originally produced validation results in the 90's, they were trained again with the expanded dataset. In addition, a model was trained using the random forest classifier for a total of seven different models. Since this dataset is multi-class, a support vector machine model could not be trained using TPOT at this time. Table 19 lists the machine learning models trained in this study. As was done in the last study, each model will be trained for 100 generations with a population size of 100 to ensure that a large number of pipelines are tested. The random state for these base models was set at 0. In order to train the models in a timely manner, High Performance Computing (HPC) resources from the INL were used. This will allow up to 48 CPUs to be utilized during model training, which should greatly improve the computation time

needed to train the models.

Table 19: Machine Learning Techniques Used to Train Models With Expanded Dataset

Gaussian Naïve Bayes	Bernoulli Naïve Bayes
Multinomial Naïve Bayes	Logistic Regression
K-nearest Neighbors	Decision Tree Classification
Random Forest Classification	

4.2.5 Validation of Trained Models

In the previous study, models were scored on four validation results based on the number of true positives, false positives and false negatives. These were: accuracy, precision, recall and f1 Score. These measurements were again used for scoring all seven models trained on the expanded dataset. This also allowed for confusion matrices to be created for each model. This will allow for individual accuracies to be measured for each transient to help address concerns with the unbalanced dataset.

4.2.6 Misclassification Analysis

Many studies in machine learning put a strong emphasis on validation results, such as accuracy and precision. These results however, should only be the start of examining a model's potential to be applied in practice, especially in a multi-class classification model. In this study, the misclassified points of the model trained were examined to determine where in the simulation the misclassifications occurred and see if there were any significant patterns. This was done by using the simulation time stamp for each data point used in the model validation. The time column removed during the model training was re-added to the data set and each point was checked to see if it was correctly identified or not. The number of misclassifications that occurred at each time during the simulation will be tracked. Using this information, it can be determined at which points during a transient event the model will be most reliable.

4.2.7 Impact of Losing Features

Due to high safety expectations by regulatory agencies and the general public, nuclear power systems require a great deal of redundancy to help mitigate possible accidents. The TMI accident showed that missing critical information and having sensor malfunctions can lead to serious errors and damaging consequences. Today, operators are trained to account for these possibilities and high quality assurance programs have been enacted. This has greatly reduced the probability of a misdiagnosis due to a sensor failure. However, if machine learning is to be implemented as a diagnostic tool, especially in an automated fashion, studies will need to be conducted on how models will be affected when features are lost. This is especially concerning if model relies on tens or hundreds of features, as many neural network models used in other applications do. One approach to address this concern is to remove features from a proposed model and train new models with fewer features. If the model can still produce reliable results with fewer features, then a system designed to help diagnose issues with the reactor, can simply switch to a new model if a feature becomes unavailable or unreliable. To contribute to this effort, this study took the models with the highest validation results and removed a number of key features to see if TPOT can train a reliable model with the feature missing.

In machine learning, it is likely some features will have more importance to the model than others. In the previous study, the decision tree model using Gini Impurity was one of the most reliable models produced. Assuming that this holds true for the expanded study, the optimal decision tree diagram will be extracted from the TPOT pipeline and the feature with the lowest impurity, i.e. the feature at the top of the tree, will be removed from the set and the model retrained. If TPOT is able to produce a reliable model, this process will be repeated for the five features with the lowest Gini Impurity in the optimal model. Another approach is to use scikit-learn to help identify possible key features.

Scikit-learn has developed a number of functions that can be used to determine feature importance for a number of different models, including tree based models, such as

decision trees and random forests. This process uses Gini Importance to assign a score to each feature used in the model training. According the documentation for scikit-learn's Decision Tree Classifier, Gini Importance is the total reduction of the criterion brought by that feature or simply the mean decrease impurity. This is calculated by summing over the number of splits in each branch of the tree[77]. This could be applied to all tree based models. As with the Gini Impurity, the feature with the highest importance score will be removed from the dataset and the model retrained. The process will be repeated for the top five individual features that have the highest importance.

4.2.8 Variation in Results from Changes in Random State

In order for effective machine learning models to be trained, there usually is a reliance on random numbers. Data sets should be split between testing and training randomly to ensure as little bias in the model as possible. This does leave open the possibility that the results from the training are a result of a favorable split. To address this concern, cross validation was used in the model training. The cross validation function within both scikit-learn and TPOT, creates different divisions of the training dataset, known as k-folds or simply folds. One fold is set aside for validation purposes and the other folds are used in the model training. The number of folds used is determined by the user. Once completed, this process is repeated again using a different fold, until all the folds have been used in validation. An average of all the results is then taken and used as the training model[78]. This technique not only address the concerns with randomly split data, but also helps address the possibility of the model being overfit.

Data splitting is not the only area that makes use of random states. Many machine learning models make use of random numbers. This includes the process used by the TPOT classifier to create pipelines. Both scikit-learn and TPOT make use of random number generators or random states in model training. In order to view the effect changes in the random state have on the trained TPOT models for this study, it was necessary to

train the models using different random states. To ensure the impact is determined, this was done with twenty different random states for all models that achieve high validation scores in the initial model training. Once completed, the variation in the results were analyzed using descriptive statistics. This will provide an idea on how each different model type is impacted by the change in random state. If the variation is small, then the model can be considered reliable in terms of random state.

4.3 Expanded Dataset Model Results

In this part of the project seven models were trained on the expanded dataset using TPOT. This section will go over the results from these models and compare them to the results from the previous section. Also, the results on the misclassification analysis will be given in this section.

4.3.1 Bernoulli Naive Bayes Models

The three naive Bayes models trained on the expanded dataset experienced significant differences from the models trained in the first part of this project. Of the three naive Bayes models trained using the expanded dataset, the Bernulli naive Bayes models scored the highest in validation results. Accuracy was scored at 81.01%, precision at 80.64%, recall at 83.24% and F1 score at 80.19%. This is much lower than the 96% 97% scores that were obtained from the original dataset. The biggest contributors to this decrease were two transients: the feedwater pump transient and the turbine trip without scram. This model only score an individual accuracy of 58.66% with the feedwater pump transient with an individual accuracy of only 58.77%. The model performed even more poorly with the individual accuracy of only 11..31%.It should be noted that this model was still able to perfectly distinguish a normally operating reactor from one experiencing a transient. Also, the model scored well with the load rejection transient when compared to the other models with an accuracy at 71.23%. Table 20 summarize the individual accuracies

for this models. Figure 36 shows the confusion matrix for this model.

Table 20: Bernoulli Naïve Bayes Model Individual Accurcies For Expanded Dataset

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	58.77%
LOCA + LOOP	98.37%
Valve Closure	87.99%
Rapid Power Change	100%
Depressurization	95.00%
Max Steam Line Rupture	99.89%
Manual Trip	83.81%
Load Rejection	71.23%
Single Coolant Pump Trip	78.78%
Total Coolant Pump Trip	82.54%
Turbine Trip without SCRAM	11.31%

Figure 36: Confusion Matrix for Expanded Bernoulli Naive Bayes Model

In terms of where misclassifications occur, the largest number of misclassifications occurred near the start of the transient event, with more than 60 misclassfications occurring at each of the first few seconds of the simulations. This number decreases to around 20 after the first half of the simulation. Figure 37 shows at what point in the simulations misclassfications occurred.

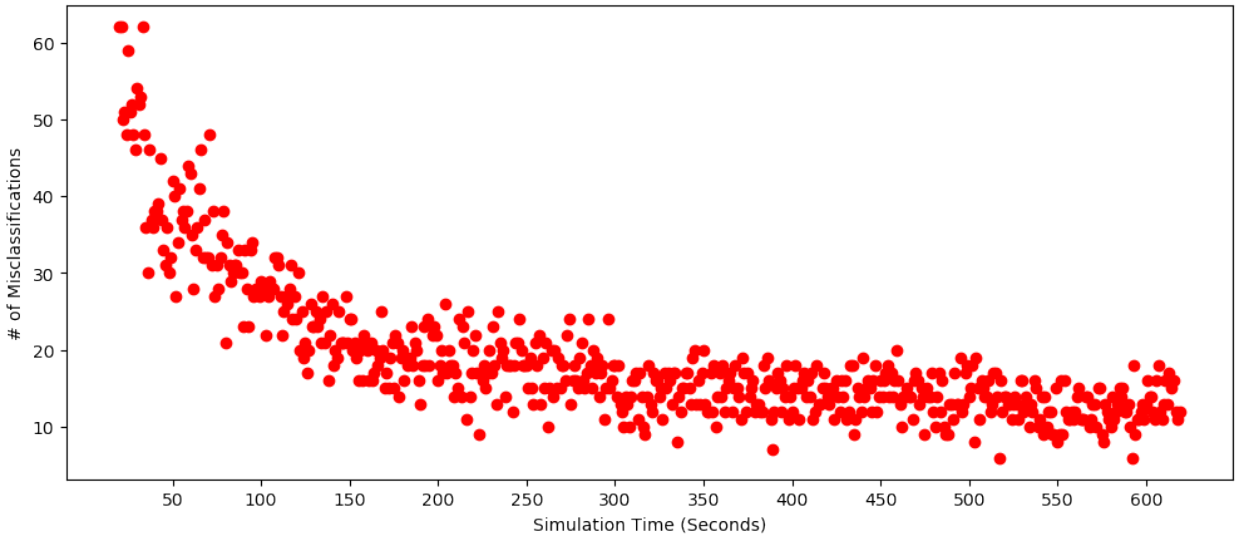


Figure 37: Misclassifications for Expanded Bernoulli Naive Bayes Model

4.3.2 Multinominal Naive Bayes Model

The multinominal naive Bayes model performed similarly to the Bernulli model. This model scored an overall accuracy of 79.60%, a precision of 79.19%, recall of 81.83% and F1- score of 79.19%. The higher recall implies that this models leans more toward false positives classifications rather than false negatives. The multinominal models struggles in the same areas as the Bernoulli model in both, the feedwater pump and turbine trip without SCRAM transients. However, this model did perform better in turbine trip without SCRAM transient with an accuracy of 19.56%. This models however, struggled much more with the single turbine trip transient than all other models trained with this dataset with an accuracy of only 67.18%. Figure 38 shows the confusion matrix for this models and Table 21 summarizes the individual validation results.

Figure 38: Confusion Matrix for Expanded Multinominal Naive Bayes Model

Table 21: Multinominal Naïve Bayes Individual Accurcies For Expanded Dataset

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	57.86%
LOCA + LOOP	98.00%
Valve Closure	81.75%
Rapid Power Change	100%
Depressurization	97.97%
Max Steam Line Rupture	99.88%
Manual Trip	77.78%
Load Rejection	67.18%
Single Coolant Pump Trip	64.30%
Total Coolant Pump Trip	85.70%
Turbine Trip without SCRAM	19.56%

The behavior for the misclassifications for this model were similar to those with the Bernoulli models. The number of misclassfications occurring at a single point again starts above 60 during the beginning of the simulations and within a 3 minutes of the simulations, decreases down to around 20. Figure 39 shows the misclassfication behavior for the multinominal model.

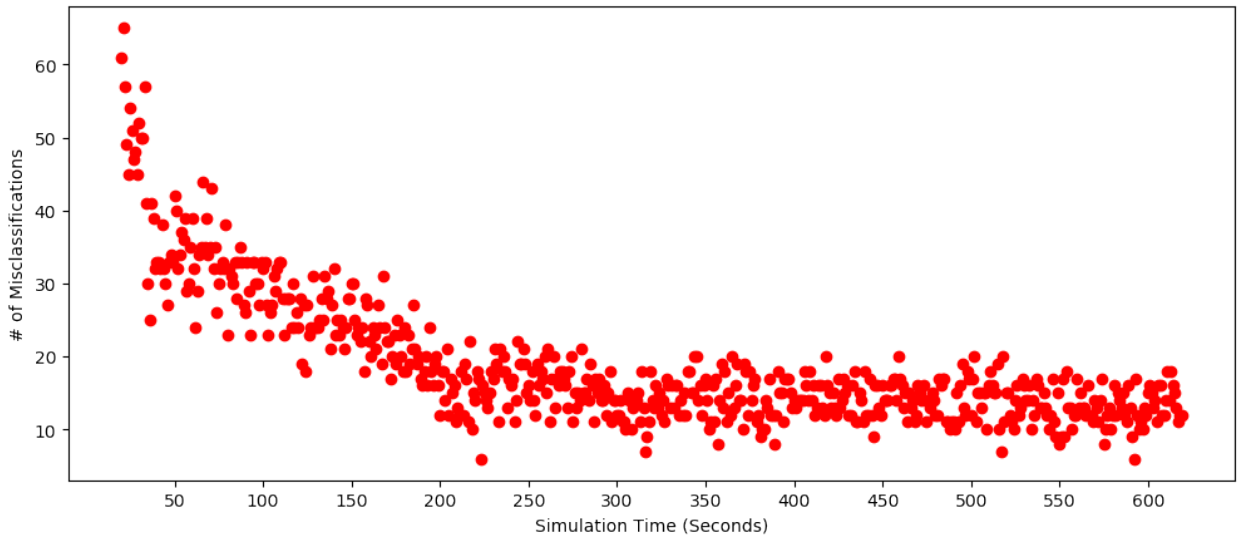


Figure 39: Misclassifications for Multinominal Naive Bayes Model

4.3.3 Gaussian Naive Bayes

The Gaussian naive Bayes model was not only the worst performing model of the three naive Bayes models, but also the worst performing model trained for this part of the project. The models scored an overall accuracy of 77.92%, a precision of 77.44%, a recall of 80.65% and a F1-score of 76.98%. As was the case with the other naive Bayes models, this model tends to experience more false positive misclassifications than false negatives. The Gaussian model was the only model trained in this experiment that was unable to perfectly distinguish the normally operating reactor from the transient events. Though it should be noted only 22 points of nearly 10,000 points tested were misclassified for this. The model struggles with the same two transients that the other naive Bayes models struggled with, but it is also hurt by issues diagnosing the load rejection, 51.97% and the total coolant pump trip, 64.30%. Both of these results are lows for this experiment. The model does have the best performance with the max steam line rupture performance, 100%, and the single coolant pump trip, 82.92%. Table 22 shows the individual accuracies for this model. Figure 40 shows the confusion matrix for this model. In terms of misclassfication behavior, the Gaussian model behaves very similarly to the other two naive Bayes models, with high numbers of misclassifications at the beginning of the simulations which decrease and level off to about 20 within a couple of minutes. It should be noted, that with this model, the first two three seconds have a low number of misclassifications compared to the rest. The reason behind this is likely a result of the model using more points for early in the simulations, as the model resumes the same behavior as the rest of the models afterwards. There is a good chance these may also be some of the normal operating points that were misclassified by the model. Figure 41 shows the misclassification behavior for the Gaussian model.

Figure 40: Confusion Matrix for Expanded Gaussian Naive Bayes Model

Table 22: Individual Accurcies For Gaussian Naïve Bayes Mode (Expanded Dataset)

Transient Event	Model Accuracy
Normal Operations	99.82%
Feedwater Pump Trip	51.81%
LOCA + LOOP	99.41%
Valve Closure	83.17%
Rapid Power Change	100%
Depressurization	97.62%
Max Steam Line Rupture	100%
Manual Trip	80.29%
Load Rejection	51.97%
Single Coolant Pump Trip	82.92%
Total Coolant Pump Trip	66.25%
Turbine Trip without SCRAM	15.93%

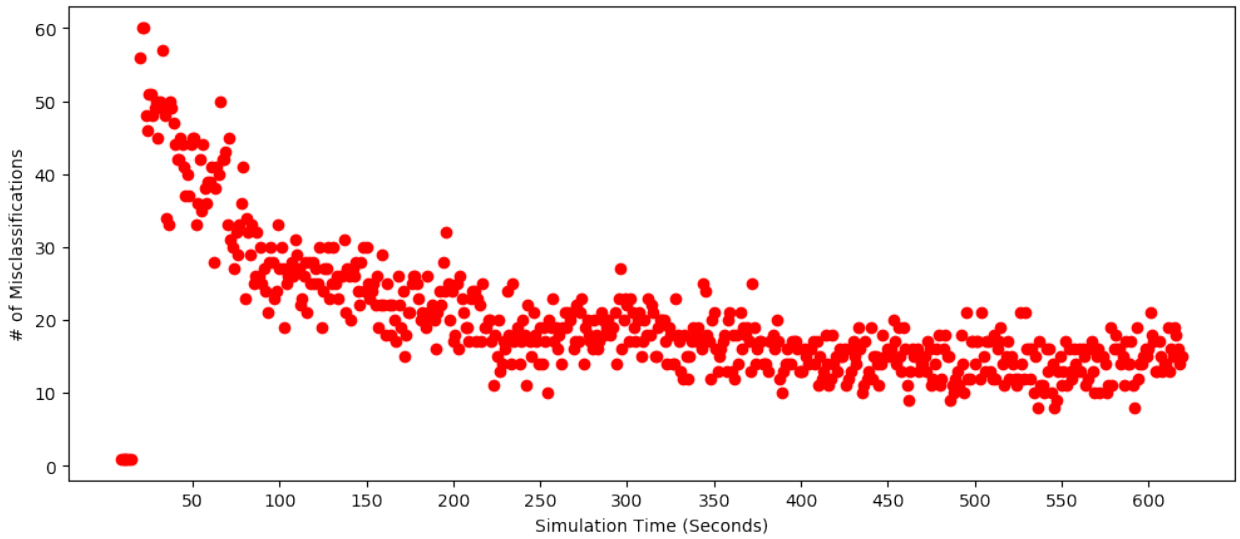


Figure 41: Misclassifications for Gaussian Naive Bayes Model

4.3.4 Logistic Regression Model

In the previous part of this project, the logistic regression model was among the highest performing models trained with overall validation results, all above 98% and individual accuracies also all above 98%. However, with the expanded dataset the logistic regression model failed to perform as strongly. In overall validation results, this model scored only 86.83% in accuracy, 86.27% in precision, 85.99% in recall and 85.85% for f1 score. This was unexpected as the model had performed strongly with the initial dataset. The main issue for this model, as is the case with the other models, is the Turbine Trip without SCRAM transient. The logistic regression model only scored individual accuracy of 29.11%. While this is better than the three naive Bayes models, it is still quite low. The model also struggles with the load rejection transient, with an accuracy of only 50.64%. The remaining transients scored well, all above 80%. The logistic regression performed better than all other models with the feedwater pump transient, with an accuracy of 80.25% and was able to perfectly diagnosis the max steam line rupture transient. Table 23 summarizes the individual transient accuracies for the logistic regression model and Figure 42 shows the confusion matrix for this model.

Table 23: Individual Accuracies For Logistic Regression Model (lExpanded Dataset)

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	80.25%
LOCA + LOOP	99.97%
Valve Closure	97.86%
Rapid Power Change	98.60%
Depressurization	97.84%
Max Steam Line Rupture	100%
Manual Trip	85.58%
Load Rejection	50.64%
Single Coolant Pump Trip	97.34%
Total Coolant Pump Trip	96.20%
Turbine Trip without SCRAM	29.11%

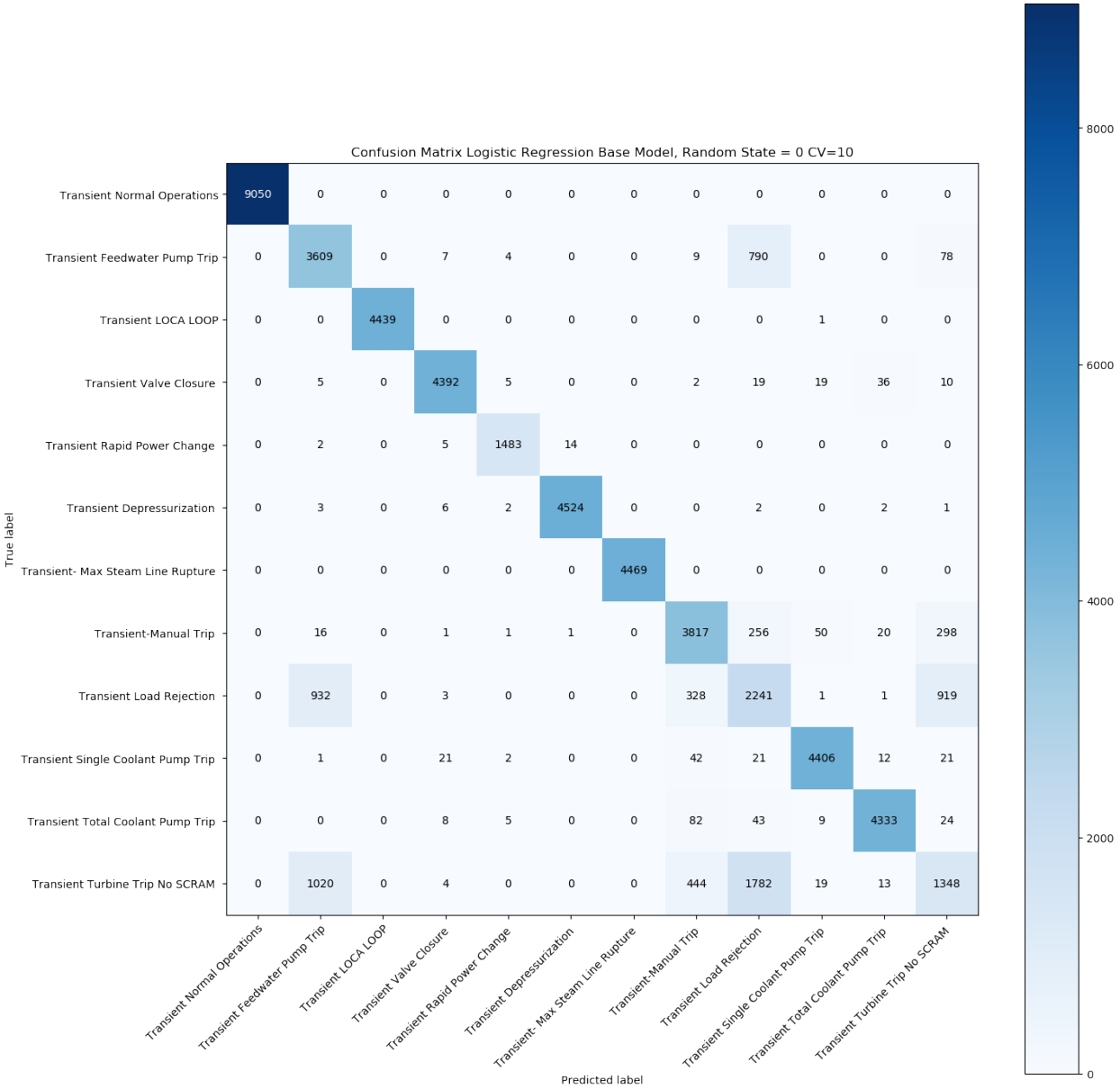


Figure 42: Confusion Matrix for Expanded Logistic Regression Model

The misclassication for the logistic regression model behave similarly to those for the other models. However, unlike the naive Bayes models, the misclassifications reduced much more quickly. In the case of this model, the number of misclassifications decreased consistently below 20 after approximately 100 seconds. It should be noted that the points at the end of the graph are a result of misclassifications occurring later within the rapid power transient. This graph is shown in Figure 43

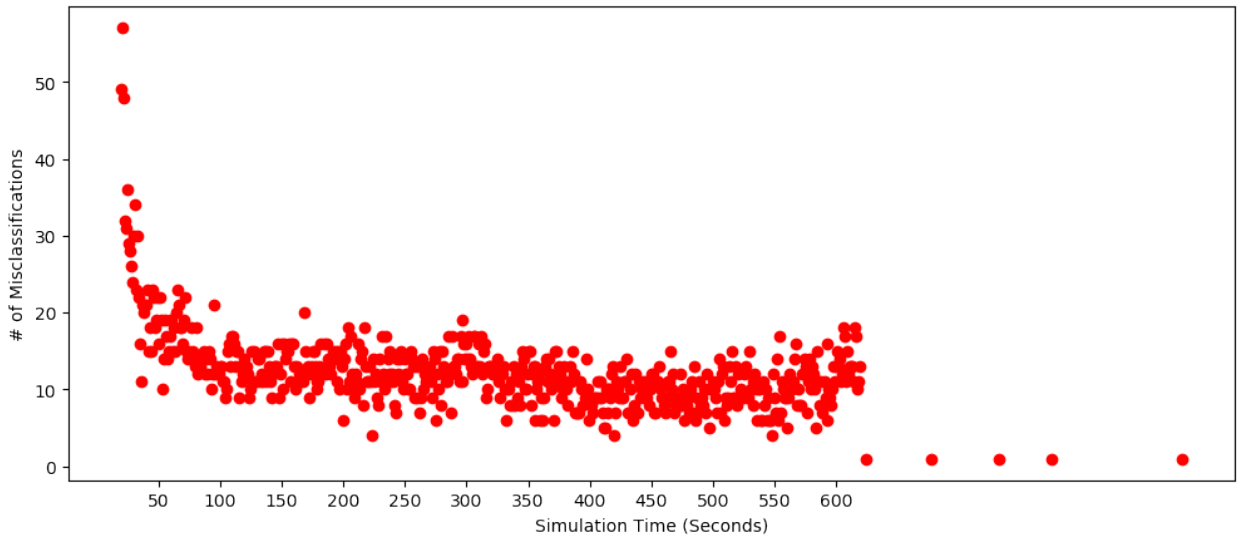


Figure 43: Misclassifications for Logistic Regression Model

4.3.5 K-Nearest Neighbors Model

During the initial phase of this project, the k-nearest neighbors models had produced some of the strongest results with the original dataset. The technique was able to once again produce strong results when used with the expanded datasets. This approach produced the best model from a non-tree based machine learning model for this experiment. In terms of overall validation results, the kNN model scored an accuracy of 91.15%, a precision of 90.93%, recall of 90.97% and f1 score of 90.91%. While this was a decrease from the initial dataset, the results are still strong. The kNN model struggled most with the load rejection models, only scoring an accuracy of 59.73%, one of the weakest scores with this transient. The model also struggled with the Turbine Trip without Scram with

an accuracy of 65.29%, but this was much higher than the four probability based models. With the exception of these two transients and the feedwater pump transient, the models was almost able to perfectly predict the remaining eight transients, with accuracies between 98% and 100%. Table 24 summarizes the individual accuracies for this model and Figure 44 shows the confusion matrix.

The kNN model produced one of the strongest results for this phase of the project and as such, it was used extensively in the other parts of this phase. Due to this, it was important to understand to what extent the model was under/overfit. To determine this, the model was tested using the training data. The accuracy of the model when tested on the training data was 93.12%. Less than 2% higher than with the testing models, as such there are few concerns with overfit. This was mostly due to issues with the feedwater pump transient, individual training accuracy of 87.9%.

Table 24: Individual Accuracies For kNN Model (Expanded Dataset)

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	73.98%
LOCA + LOOP	99.99%
Valve Closure	98.37%
Rapid Power Change	100%
Depressurization	99.66%
Max Steam Line Rupture	99.97%
Manual Trip	97.42%
Load Rejection	59.73%
Single Coolant Pump Trip	98.45%
Total Coolant Pump Trip	98.31%
Turbine Trip without SCRAM	65.29%

In the area of misclassification behavior, the kNN model behaved similarly to the four probability based models, where there are a relatively large number of misclassifications at the start of the simulations and the number decreases as the simulation continues. In the case of the kNN model, the number of misclassifications drops to around 20 after about 10 seconds. Figure 45 shows the misclassification behavior for this model.

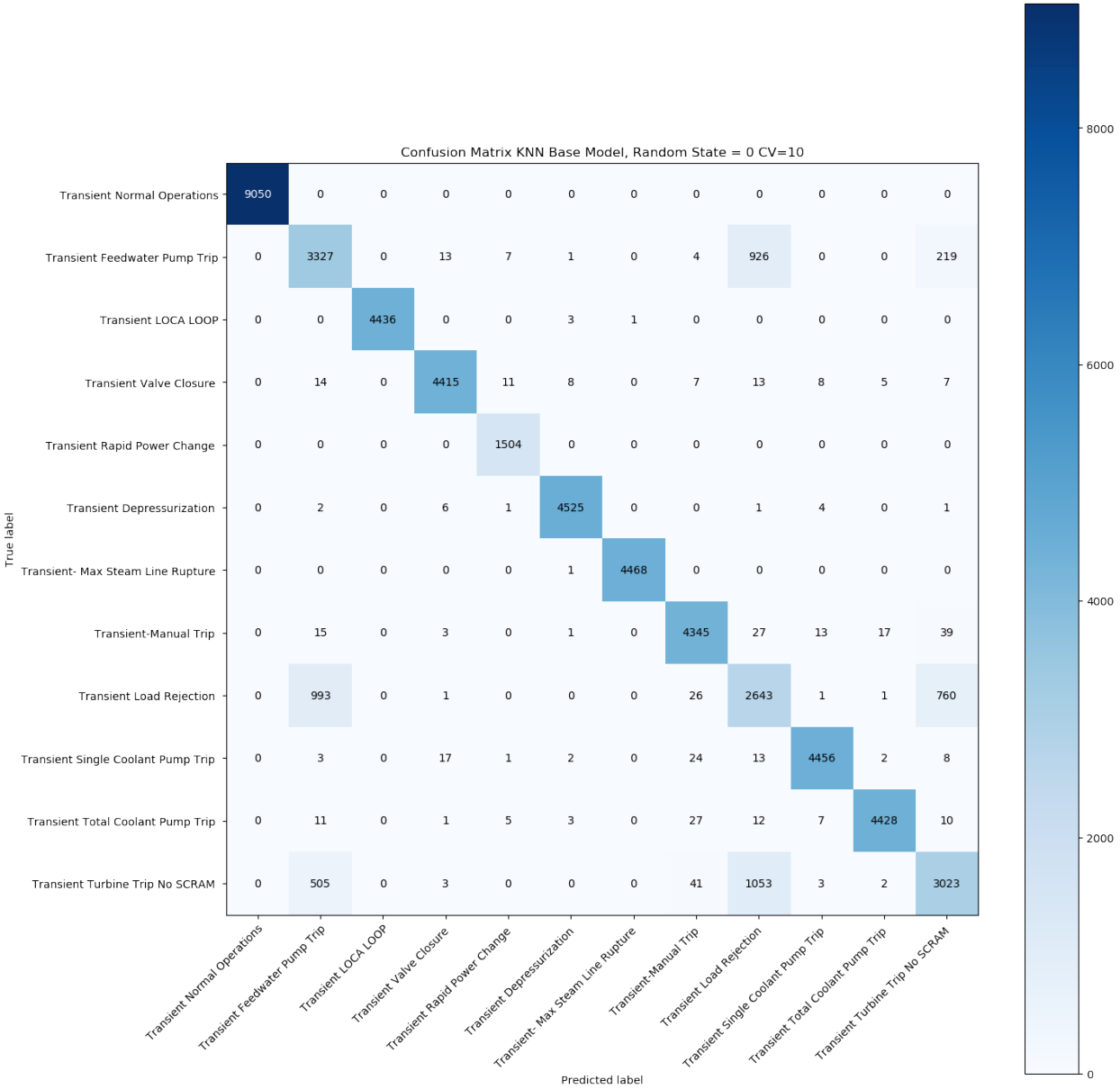


Figure 44: Confusion Matrix for Expanded kNN Model

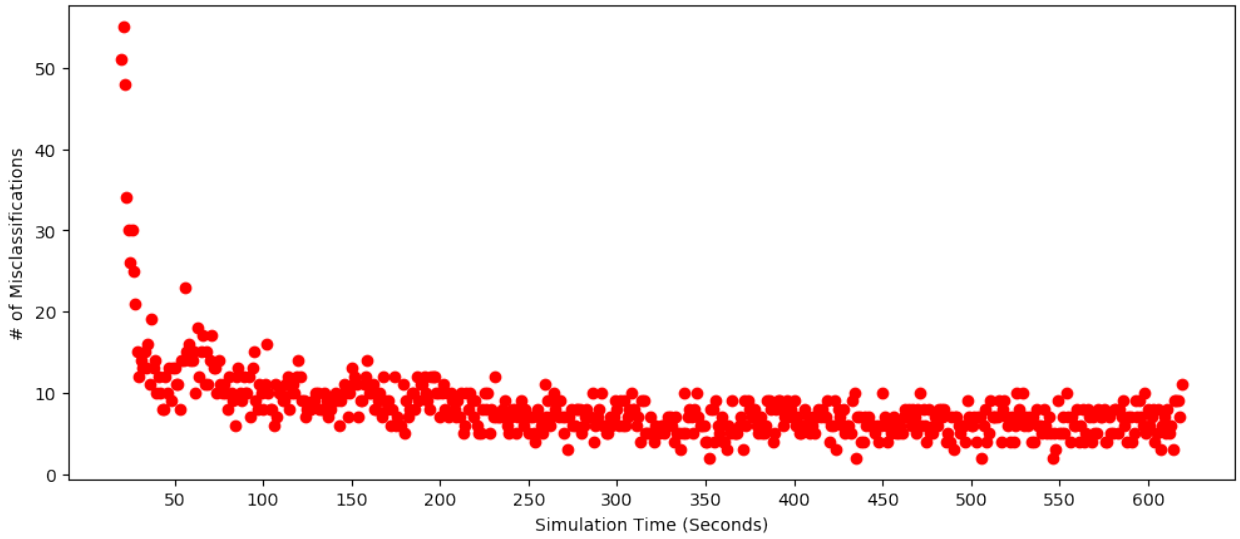


Figure 45: Misclassification Behavior for kNN Model

4.3.6 Decision Tree Model

In the previous part of this project, the decision tree model trained using TPOT was the best performing model of the six models trained. When the decision tree model was trained using the expanded dataset, the model once again proved to be one of the best performing. The overall accuracy of this model was 92.38%, with a precision and recall of 92.15% and a f1 score of 92.14%. In terms of individual accuracy, the model performed strongly with all but three of the transients. The three lower performing transients were: feedwater pump transient, accuracy of 72.98%, turbine trip without SCRAM, 74.71% and the load rejection, 63.48%. While these were significantly lower than the other transients, these were among some of the highest scores with these three transients. Figure 46 shows the confusion matrix for this model and Table 25 summarizes this model's individual accuracies. Since this model was one the best performing models in both parts of the project, it was necessary to evaluate how overfit the model is by comparing the training and testing accuracies of the model. The training accuracy was found to be 99.68%, a little more than six points higher testing accuracy. This is mostly a result of the three poorer performing transients. This does indicate the model is notably overfit. It is important to

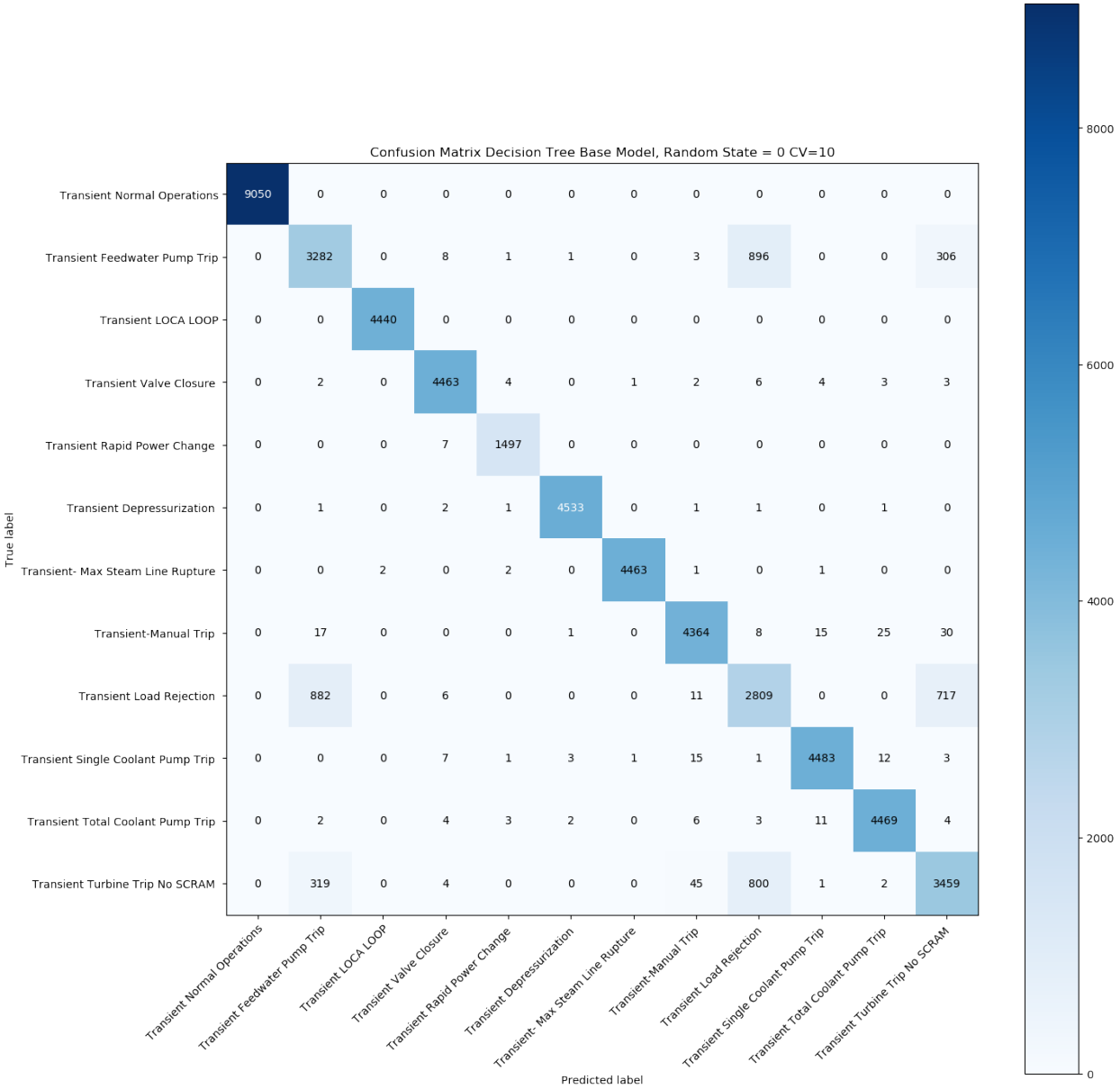


Figure 46: Confusion Matrix for Expanded Decision Tree Model

Table 25: Individual Accuracies For Decision Tree Model (Expanded Dataset)

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	72.98%
LOCA + LOOP	100%
Valve Closure	99.44%
Rapid Power Change	99.53%
Depressurization	99.84%
Max Steam Line Rupture	97.84%
Manual Trip	97.84%
Load Rejection	63.48%
Single Coolant Pump Trip	99.92%
Total Coolant Pump Trip	99.05%
Turbine Trip without SCRAM	74.71%

note that most tree based models experience a degree of overfit, and this amount while notable, does not disqualify the usefulness of the model. In the area of misclassification behavior, the decision tree model behaved similar to the kNN model. The model experiences around 50 misclassifications in the first second of the transient simulations. This number decreases below 20 at around the seven second mark. The number stays around 10 for the majority of the time across all simulations. Figure 47 shows a graph of the misclassification behavior for this model.

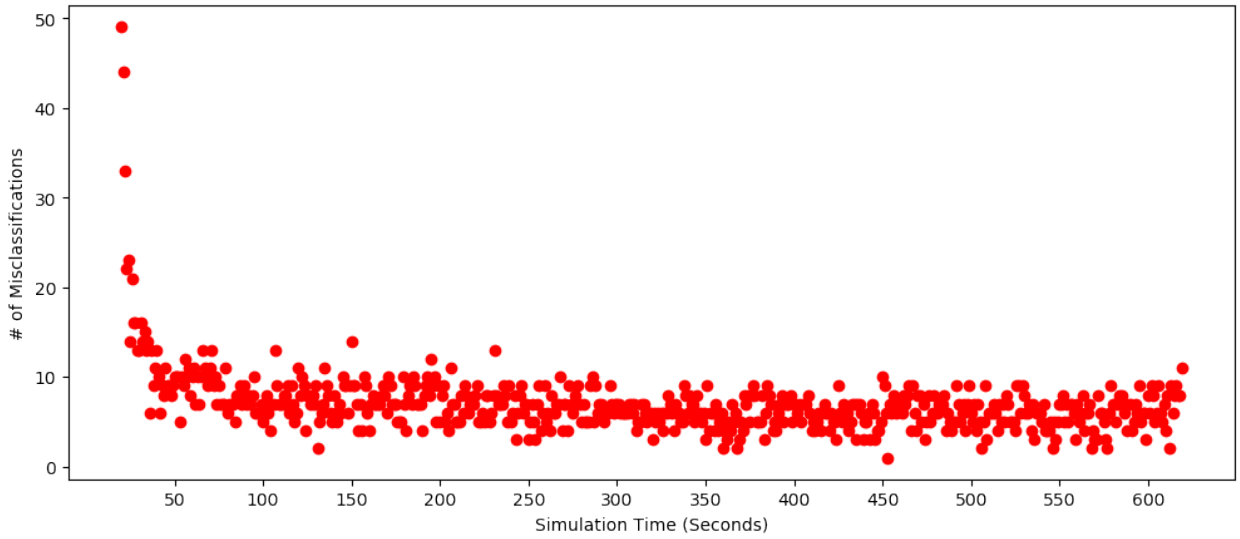


Figure 47: Misclassifications for Decision Tree Model

4.3.7 Random Forest Model

At the time of the first phase of this project, TPOT had just begun to implement support for scikit-learn's packages on support vector machines and random forests. As such, it was not possible to train a random forest model for the initial dataset. Using the expanded dataset, a random forest model was trained with TPOT. This resulted in the best performing model of the seven trained for this phase of the project. The model scored an accuracy of 92.44%, recall of 92.23% and a precision and f1 score of 92.22%. All slightly higher than the single decision tree model. Individual accuracies were also similar to that of the decision tree model. The model scored high with the eight transient events that the decision tree model scored high. The model also struggles most with the same three transients, with an individual accuracy of 75.5% for the turbine trip without SCRAM, slightly higher than the decision model, 70.98% for the feedwater pump transient and 62.89% for the load rejection transient, both slightly lower than the decision tree model. Figure 48 shows the confusion matrix for this model and Table 26 summaries the model's individual accuracies. Table 27 shows the accuracies for the first 30 seconds of the simulation for the decision tree model.

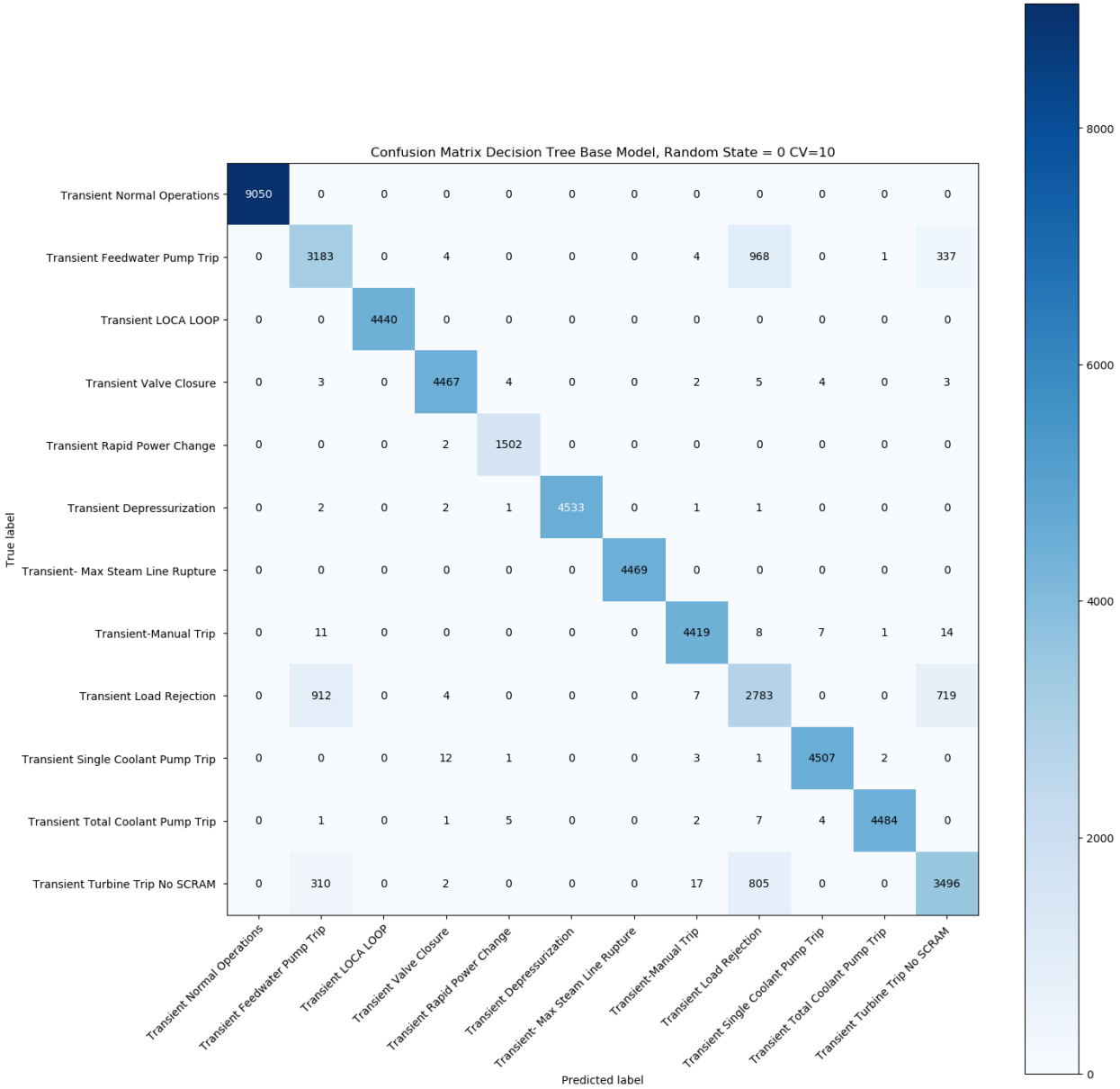


Figure 48: Confusion Matrix for Expanded Random Forest Model

Table 26: Individual Accuracies For Random Forest Model (Expanded Dataset)

Transient Event	Model Accuracy
Normal Operations	100%
Feedwater Pump Trip	70.78%
LOCA + LOOP	100%
Valve Closure	99.53%
Rapid Power Change	99.86%
Depressurization	99.84%
Max Steam Line Rupture	100%
Manual Trip	99.08%
Load Rejection	62.89%
Single Coolant Pump Trip	99.58%
Total Coolant Pump Trip	99.55%
Turbine Trip without SCRAM	75.50%

As was done with both, the single decision tree and the kNN model, it was necessary to check this model for overfit, as it was one of the better performing models. The random forest model's training data scored a 99.68%. This is similar to the decision tree indicating that this model does experience some overfit once again, in the area of the three problematic transient events. It is important to note then, that while this model does experience some notable overfit, this is expected due to the nature of tree based models.

In terms of misclassifications, the random forest model experienced the same behavior of the single decision tree model. The model experienced around 45 misclassifications at the first second of the transient simulations and this number dropped quickly below 20 within 5 seconds. Throughout most the remaining time of the simulations, the model scored fewer than 10 misclassfications at each second. Figure 49 shows the behavior for the random forest model.

Table 27: Accuracies of Decision Tree Model During The First 30 Seconds

Time	Counts Tested	Counts Misclassified	Accuracy
0	76	47	38.16%
1	85	49	42.35%
2	88	33	62.5%
3	74	36	64.86%
4	78	23	70.51%
5	88	12	86.36%
6	85	22	74.11%
7	78	14	82.05%
8	82	15	81.70%
9	85	13	84.71%
10	79	13	82.28%
11	91	16	82.41%
12	88	20	77.27%
13	93	15	83.87%
14	89	13	85.39%
15	73	14	80.82%
16	69	5	92.75%
17	91	13	85.39%
18	94	11	88.29%
19	81	11	86.41%
20	79	12	84.81%
21	79	12	84.81%
22	88	7	92.05%
23	86	9	89.54%
24	85	10	88.23%
25	86	9	86.20%
26	85	10	89.41%
27	87	12	85.89%
28	85	12	89.41%
29	82	5	93.90%
30	87	9	89.65%

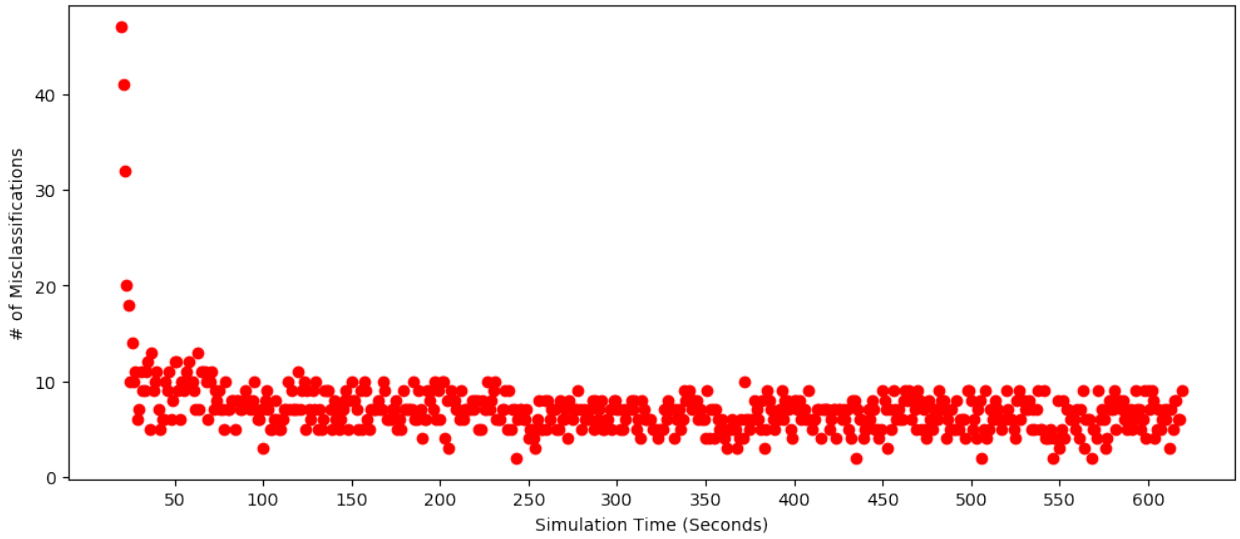


Figure 49: Misclassifications for Random Forest Model

4.4 Random State Variation Analysis

The decision tree model has been one of the best performing models throughout both, this study and the previous experiment. The model experienced minimal variation in terms of overall validation results when the random state was changed. Of the twenty models trained, the highest accuracy score was 93.09% and the lowest was 92.30%. The median accuracy between the twenty decision tree models was 92.41%, close to the accuracy reported for the base model. This trend is similar for the precision, recall and f1 scores, with median scores of 92.18% for precision and 92.17% for recall and f1 score. This shows for the decision tree model, there is little variation between different random states. Table 28 summarizes the statistics for the twenty decision tree models.

Table 28: Statistics for Decision Tree Variation Analysis

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Mean	92.45%	92.22%	92.23%	92.22%
Median	92.41%	92.18%	92.17%	92.18%
Standard Deviation	0.16%	0.17%	0.18%	0.15%
Minimum	92.30%	92.06%	92.07%	92.07%
Maximum	93.09%	92.90%	92.94%	92.83%
Range	0.79%	0.84%	0.87%	0.76%

The random forest model has also produced high results in this experiment. Similar to the decision tree, the random forest model also experienced minimal variation between the twenty different models. The highest trained accuracy for the random forest models was 92.48% and the lowest was 92.34%. The median accuracy between these models was 92.42%, again very close to the 92.41% scored for the base model. The median precision for these models was 92.20%, as was the median f1 score. The median recall was 92.21%. The random forest models showed less variation than the decision tree models, with a range of accuracy less than 0.14%. Table 29 shows the statistics for the random forest models.

Table 29: Statistics for Random Forest Variation Analysis

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Mean	92.42%	92.20%	92.21%	92.20%
Median	92.42%	92.21%	92.22%	92.21%
Standard Deviation	0.03%	0.03%	0.03%	0.03%
Minimum	92.34%	92.12%	92.11%	92.27%
Maximum	92.48%	92.26%	92.28%	92.27%
Range	0.14%	0.14%	0.16%	0.15%

Although the k-nearest neighbors models was not as accurate as the two tree based models, the base model produced high results in the low 90's. This model was also the least overfit of the three best models trained in this phase of the project. The twenty k-nearest neighbors experienced more variation than the tree based models. The median accuracy of 91.42% is slightly lower than the accuracy of the base model at 91.82%. The other three validation measurements are similar with a median precision of 91.20%, recall of 91.08% and f1 score of 91.01%. The range of accuracies measured was 1.08%, slightly larger than the range of the two tree based models.The statistics for the k-nearest neighbors models are summarized in Table 30.

Table 30: Statistics for K-Nearest Neighbors Variation Analysis

Model	Accuracy	Precision	Recall	F1 Score
Mean	91.27%	91.05%	91.14%	91.01%
Median	91.24%	91.01%	91.05%	91.98%
Standard Deviation	0.69%	0.71%	0.74%	0.72%
Minimum	88.74%	88.48%	88.53%	88.33%
Maximum	92.06%	92.18%	91.99%	91.80%
Range	3.32%	3.34%	3.46%	3.47%

4.5 Improving the Model

Once a machine learning model has been trained, it is common practice to look for ways to improve the model. This is especially true if weaknesses within the model can be identified. In this case, the three best performing models all struggled with three of the 12 different transient events classified on. This section will explore different methods used to improve these models. Two different approaches were used for this part of the experiment. The first was changing the split between testing and training data to see if more training data can improve model performance. The second approach is to increase the number of generations and population size used with the TPOT Classifier. It is important to note that there are other approaches that could be used to improve a machine learning model's performance.In many cases, the first strategy to improve a machine learning model is to better tune hyperparameters by trying different combinations to see if a better model can be produced. In machine learning hyperparameters are parameters that cannot be changed in the training process. Some common examples of hyper parameters include, the number nearest neighbors in a kNN model, the max depth of a tree based model, the number of neurons used in a layer in training a neural network and the activation function used in training a neural network model. Tuning can be a complex process and their are a number of different approaches that can be used such as Grid Search, gradient boosting, etc. However, in the case of this project an AutoML package is

used to train the machine learning models. One of the benefits of AutoML packages like TPOT is that the package will try several different hyperparameter combinations. While this is a brute force approach and is time consuming, it is effective in producing better performing models. The use of HPC makes the use of AutoML feasible in this case. Due to this approach, there is no need for further hyperparameter tuning. Another common approach to try and improve a machine learning model's performance is to add more data to the dataset. This can be done in two different ways. The first is to collect more samples and increase the size of the dataset. Since the expanded dataset already consisted of over 100,000 data points and already several initial conditions had been simulated, it was decided that a better approach would be to change the split between testing and training data. As has been mentioned, the dataset was split in two for the training of the models. To see if additional data would improve the model, this split was changed from 50% testing and 50% training to 55% testing/45% training, 60% testing/40% training and 65% testing/35% training. The three optimal models were then re-trained and validated. All three models were also re-trained using a 45% testing/65% testing to see the impact that reducing the data would have on the optimal models.

The results from increasing the testing data were surprising. Instead of improving the model's accuracy and other validation measurements, the measurements actually appear to have slightly decreased as the testing data split increased. It should be noted that the drop is very small within the range of the variation shown in the previous section. For example, the decision tree model's original accuracy was 92.38%. This dropped very slightly to 92.33% with the split changed to 55% training and 45% testing. The trend continued as the amount of testing data increased. When the model was training with a 60% training split, the accuracy dropped down to 92.16%. The model's accuracy remained the same when the split was increased to 65%. The model improved very slightly to an accuracy of 92.43% when the split was decreased to 45%. Table 31 shows the validation results for all four different splits for the decision tree. The random forest model experi-

enced a similar trend with validation measurements decreasing as data was added and increasing when training data was reduced. Table 32 summarizes the results for the random forest model. It should also be noted that the gap in validation results stayed within the same range as the original model with a 6 7% difference. The reason for the change in validation results with these tree based models is most likely the model becoming more overfit and more rigid to outlying data as more testing data is brought in. When this occurs it can become more difficult for the model to classify correctly data that has some outliers.

Table 31: Validation Results for Decision Tree Models Trained with Different Splits

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
55%-45% Split	92.33%	92.10%	92.11%	92.10%
60%-40% Split	92.16%	91.97%	91.94%	91.95%
65%-35% Split	92.16%	91.96%	91.94%	91.95%
45%-55% Split	92.43%	92.20%	92.19%	92.19%

Table 32: Validation Results for Random Forest Models Trained with Different Splits

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
55%-45% Split	92.28%	92.05%	92.07%	92.05%
60%-40% Split	92.28%	92.14%	91.94%	91.93%
65%-35% Split	92.10%	91.89%	91.88%	91.88%
45%-55% Split	92.60%	92.37%	92.39%	92.38%

The kNN model behaved differently than the two three based models. The original kNN model scored an accuracy of 91.15%. When the split was changed to 55% training data 45% testing, the accuracy of model decreased to 90.95% and then down to 90.85% when the split was changed to 60%-40%. However, when the split was changed to 65%-35% the model's accuracy went up to 92.3%, a fairly significant increase. The model's accuracy also increased slightly, to 91.69% when the split was decreased to 45%-55%. This suggests that the increase in validation results is more likely due to TPOT exploring different pipelines rather than any significant gain from having additional data or the model being overfit. It should be noted that the kNN model's validation accuracy remained only 1 2% higher than the testing accuracy through this process. Table 33 summarizes this

model.

Table 33: Validation Results for kNN Models Trained with Different Splits

Model	Accuracy	Precision	Recall	F1 Score
55%-45% Split	90.95%	90.70%	90.84%	90.70%
60%-40% Split	90.85%	90.62%	90.75%	90.62%
65%-35% Split	92.34%	92.13%	91.20%	91.14%
45%-55% Split	91.69%	91.45%	92.50%	92.45%

Another possible approach to add data to the dataset is to look to collect more features for the dataset. This is sometimes referred to as increasing the dimensionality of the dataset. By adding more features it is possible the model could better identify their target data in supervised classified learning or discover new trends in unsupervised classified learning. It should be noted there are a number of drawbacks associated with this approach. For example, adding new features will add more complexity to the dataset and likely increase the overfit of the models produced. This is known as the "Curse of Dimensionality".

There are also a number of issues with increasing the dimensionality that are specific to this experiment. The first is the time associated with collecting the data. The data collected for the expanded dataset took a number of weeks to compile through simulation. To add more features to this dataset it would likely take a number of months to identify potential features, re-run the simulations, compile the dataset and retrain the models. A more important consideration for collecting additional features is the potential to create more dependency in the model. The second approach performed in order to improve the performance of the models was increasing the number of generations and pipelines used in the training. This approach is one of the advantages of TPOT, as it allows the classifier more time to produce a better model. This primary issue is that training is already a time consuming process and increasing the population size and/or increasing the number of generations run will significantly increase the time for the training process. However, with the use of HPC it is possible to train these models in a reasonable amount of time.

The first attempt to improve the model was done by increasing the population size, number of pipelines retained in each generation from 100 to 150. This was run for 100 generations. In the next attempt, the population size was increased to 200 while running for 100 generations. The final two attempts to improve the model increased the number of generations run along with the number of pipelines. The number of generations was first increased to 150 and then again to 200. The population size for these two runs stayed at 200. This was performed for each of the three optimal models.

The initial increase in population size to 150 made little difference on all three models. The decision tree model scored an accuracy of 92.37%, about the same as the original model 92.38%. The random forest model scored a 92.46% in accuracy, slightly higher than the original model at 92.44%, but still within the range from the variation analysis. The kNN model dropped slightly to 91.00% from the original 91.15%, still within the range from the variation analysis. The results from increasing the population size to 200 were similar. The decision tree model improved up to 92.40% and the random forest model score remained at 92.46%. The kNN model experienced a more significant increase to 91.62% higher than both of the other models trained using the technique, however this value is still within the 3.32% range established in the previous section.

The attempt to improve the model by increasing the number of generations TPOT ran, also had little impact on the three model's overall performance. When the number of generations was increased to 150 with the population size remaining at 200, the decision tree model score 92.42% in accuracy, the random forest model scored 92.44% and the kNN model scored a 91.64%. All three of these results, while slightly higher than the previous attempt, were still well within the range established in the random variance section. This pattern continued when the number of generations was increased to 200. The decision tree model and random forest models dropped slightly to 92.38% and 92.42% and the kNN model increased to 91.72%. All of these values were well within the expected range. Table 34, 35 and 36 summarize all the validation results for these attempts.

Table 34: Validation Results for Decision Tree Models Trained with Different TPOT Parameters

Model	Accuracy	Precision	Recall	F1 Score
150 Population Size- 100 Generations	92.37%	92.13%	92.14%	92.13%
200 Population Size- 100 Generations	92.42%	92.17%	92.17%	92.17%
200 Population Size- 150 Generations	92.42%	92.19%	92.20%	92.19%
200 Population Size- 200 Generations	92.38%	92.15%	92.17%	92.15%

Table 35: Validation Results for Random Forest Models Trained with Different TPOT Parameters

Model	Accuracy	Precision	Recall	F1 Score
150 Population Size- 100 Generations	92.46%	92.24%	92.25%	92.24%
200 Population Size- 100 Generations	92.46%	92.25%	92.25%	92.25%
200 Population Size- 150 Generations	92.44%	92.22%	92.22%	92.22%
200 Population Size- 200 Generations	92.44%	92.20%	92.22%	92.21%

Table 36: Validation Results for kNN Models Trained with Different TPOT Parameters

Model	Accuracy	Precision	Recall	F1 Score
150 Population Size- 100 Generations	91.00%	90.76%	90.85%	90.74%
200 Population Size- 100 Generations	91.62%	91.39%	91.42%	91.37%
200 Population Size- 150 Generations	91.64%	91.41%	91.44%	91.40%
200 Population Size- 200 Generations	91.71%	91.49%	91.53%	91.47%

4.6 Identifying Reasons Behind Misclassifications

Since the efforts to improve the three optimal models didn't resolve the issues with the models, it was necessary to examine the models and try to determine what is causing the misclassifications. Of the 12 events the models attempted to classify, ninodels scored in the 70% range, while the load rejection transient scored in the 60% ragne. It should be noted that the vast majority of misclassifications occurred between these three transients, i.e. load rejections were misclassified as feedwater pump trips or turbine trips, etc. The models trained using TPOT make use of a variety of feature reduction and selection techniques. As such, it can be difficult to identify a single root cause for a transient.

To better examine the reason behind the larger number of misclassifications with these three transients, it was necessary to further examine the data used in testing the models. In order to do this, an optimal pipeline from the the variation analysis was re-trained and tested. Using Pandas, it was possible to create dataframes consisting of the data that was correctly classified for each of the three transients. These dataframes consisted of 2797 points for the load rejection transient, 3290 for the feedwater pump transient and 3463 points for the turbine trip without SCRAM. Six separate dataframes were created containing the misclassfied data for each of the transients. The descriptive statistics for each set of data were calculated and then compared to look for similarities and differences that could have resulted in misclassifications.

In examining the possible reasons behind the misclassifications, the load rejection transient's descriptive statistics show a number of significant differences between the correctly identified load rejection points and the 899 load rejection points incorrectly identified as feedwater pump trips. The most significant of these was the level of the narrow range of SG-2. In the correctly identified load rejections points, the level was found to be 63% above capacity, level 163%, while in the incorrect points, the level was found to be on average 25%. This value is much closer to the correct feedwater pump trip's average of 10%. The model also incorrectly identified 714 points load rejection as turbine trips without SCRAM. Once again, the most significant area of difference was the SG-2 NR LEVEL feature, where the incorrect load rejection points diagnosed as turbine trips without SCRAM had an average level of 127% compared to the 163% of the correct points. The correct turbine trip points had an average level of 136%. Other features with notable differences for load rejection included MS FLOW FROM SG-1 LINE-1A, 1B, 2A & 2B, as well as the FW FLOW TO SG-2.

Examining the feedwater pump transient data, there were 883 feedwater pump trip points incorrectly identified as turbine trips without SCRAM. The most notable difference was with the narrow range level of SG-1. The correct feedwater pump points had

an average level of 12%, while the misclassified points had a much higher average at 54%. The average for the correct turbine trip pumps was much closer to the incorrect points at 55%. The trend was similar for the 883 feedwater pump trip points that were incorrectly classified as load rejection. The average value for the SG-1 NR LEVEL for these points was 54%, very close to the average for the correct load rejection points at 55%. The generated power and pressure from SG-2 were also noticeably different between sets.

Finally, for the turbine trip without scram data, 331 points were misclassified as feedwater pump trips. Similar to the load rejection data, the most notable feature was the SG-2 NR LEVEL. The mean for this feature from the incorrect data was 25%, very close to the value for the correct feedwater pump transient and significantly lower from the 136% average of the correct turbine trip without SCRAM data. In the case of the turbine trip without SCRAM points that were incorrectly identified as load rejection, there appears to be no noticeable similarity between the narrow range levels of SG-1 and 2, as both correct sets of data have noticeable differences from the incorrect set. It is likely that this incorrectly classified data is the result of the data being extremely close to the data of the other transient and not the result of a single feature. Table 37 and 38 show the averages for both SG-1 and SG-2 Levels for all of the data subsets. The complete list of descriptive statistics for the subsets can be seen in the Appendix section.

Table 37: Steam Generator Level Averages for Correctly Classified Data

Transient	**SG-1 LV (%) Average**	**SG-2 LV (%) Average**
Electrical Load Rejection	53.779056	162.777024
Turbine Trip W/O SCRAM	55.364462	136.219546
Feedwater Pump Trip	12.910617	10.124434

4.7 Decision Tree Analysis

As was the case in the previous study, the decision tree model scored some of the highest results of the study. This allowed for a deeper analysis of the decision tree model and its reliance on certain features. The first step to perform this analysis was to extract the

Table 38: Steam Generator Level Averages for incorrectly Classified Data

Transient	SG-1 AVG Level(%)	SG-2 AVG Level(%)	SG-1 Difference	SG-2 Difference
Turbine Trip as Load Rejection	110.192612	54.220982	1.14348	110.418089
Turbine Trip as Feedwater Pump	54.361579	25.274029	1.002883	110.945517
Feedwater Pump as Load Rejection	54.220982	25.801457	41.310365	15.677023
Feedwater Pump as Turbine Trip	54.220982	25.801457	41.310365	15.677023
Load Rejection as Turbine Trip	2.861543	127.372561	0.605488	63.379265
Load Rejection as Feedwater Pump	54.384544	99.397759	0.408784	136.789748

decision tree from the optimal pipeline produced, using TPOT and graph the tree using scikit-learn. The tree produced from this model was quite large, due to the number of classes and features used in training. The top five levels of the tree are shown in Figure 50. It can be seen that this optimal model is able to make some quick classifications based on a single feature in the dataset. This leaves the following question: can a model be trained, that lacks one or more of these key features and still make an accurate diagnosis? To determine this, the top five features of the tree were dropped one by one and new models retrained using TPOT. Table 39 lists the features that were dropped from the dataset. The first feature dropped was the Pilot Operating Relief Valve (PORV) discharge

Table 39: Features Removed from Optimal Decision Tree (Gini Impurity)

Feature	Feature Number
PORV Discharge Temperature	15
RCS LVL Loop 1 NR	3
Cold Leg 2A Temperature	8
Hot Leg 2 Temperature	7
Hot Leg 1 Temperature	4

temperature. The model produced using the decision tree was much more complex, with a much larger tree produced. However, TPOT was still able to produce a high performing model. The accuracy of this model was scored at 92.33%, a marginal difference from the original decision tree model's 92.38%. The other validation measurements scored simi-

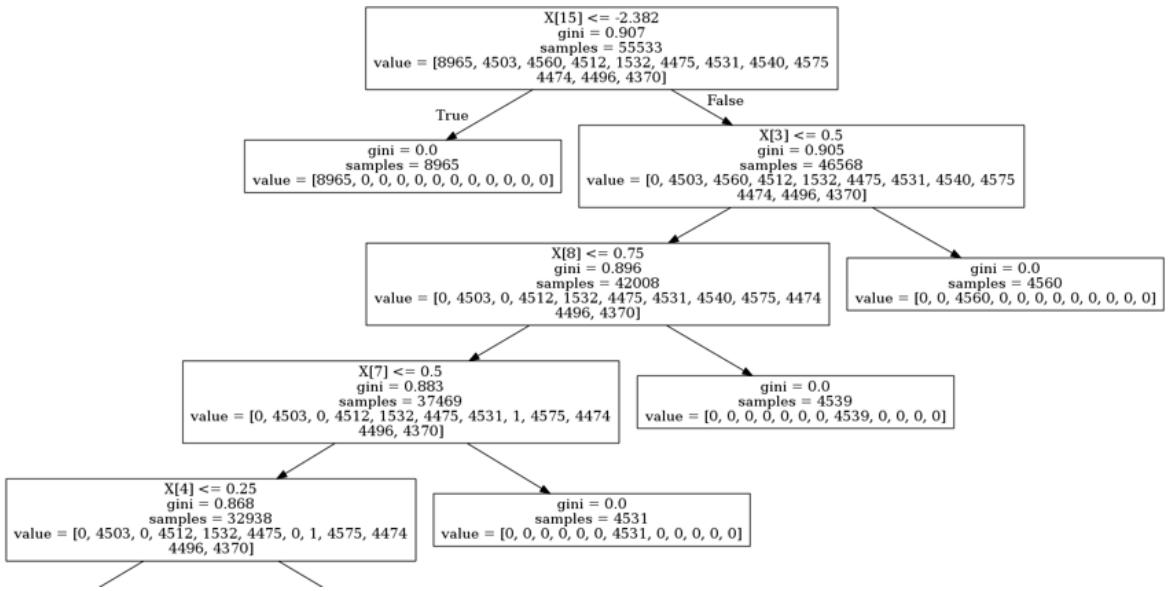


Figure 50: Top 5 Levels of Optimal Decision Tree

lar as well. The models trained using the random forest increased slightly with a score of 92.46%. The k-nearest neighbors model also increased up to 91.60% from 91.15%. This is not entirely unusual as performance can degrade in this type of model with a high number of features and there is some expected variation due to the use of random numbers.

The second feature that was removed from the dataset was the water level narrow range for RCS loop. This time, the decision tree's accuracy increased slightly to 92.40% The random forest model fell slightly to 92.39% and the k-nearest neighbors model once again, improved up to 92.42%. This trend continued when the third feature, the 2A cold leg temperature was removed, the decision tree once again improved up to 92.45% and the random forest stayed consistent at 92.39%. The k-nearest neighbors model however, fell down to 91.76%. The fourth feature dropped from the dataset was the temperature for the second hot leg of the reactor system. The decision tree model began to fall slightly when this feature was removed, scoring an accuracy of 92.36%. The random forest model slightly improved with the removal of this feature to 92.46%, while k-nearest neighbors models also scored slightly lower at 91.07%. The final feature dropped for this part of the study was the temperature for the first hot leg of the reactor system. Once again,

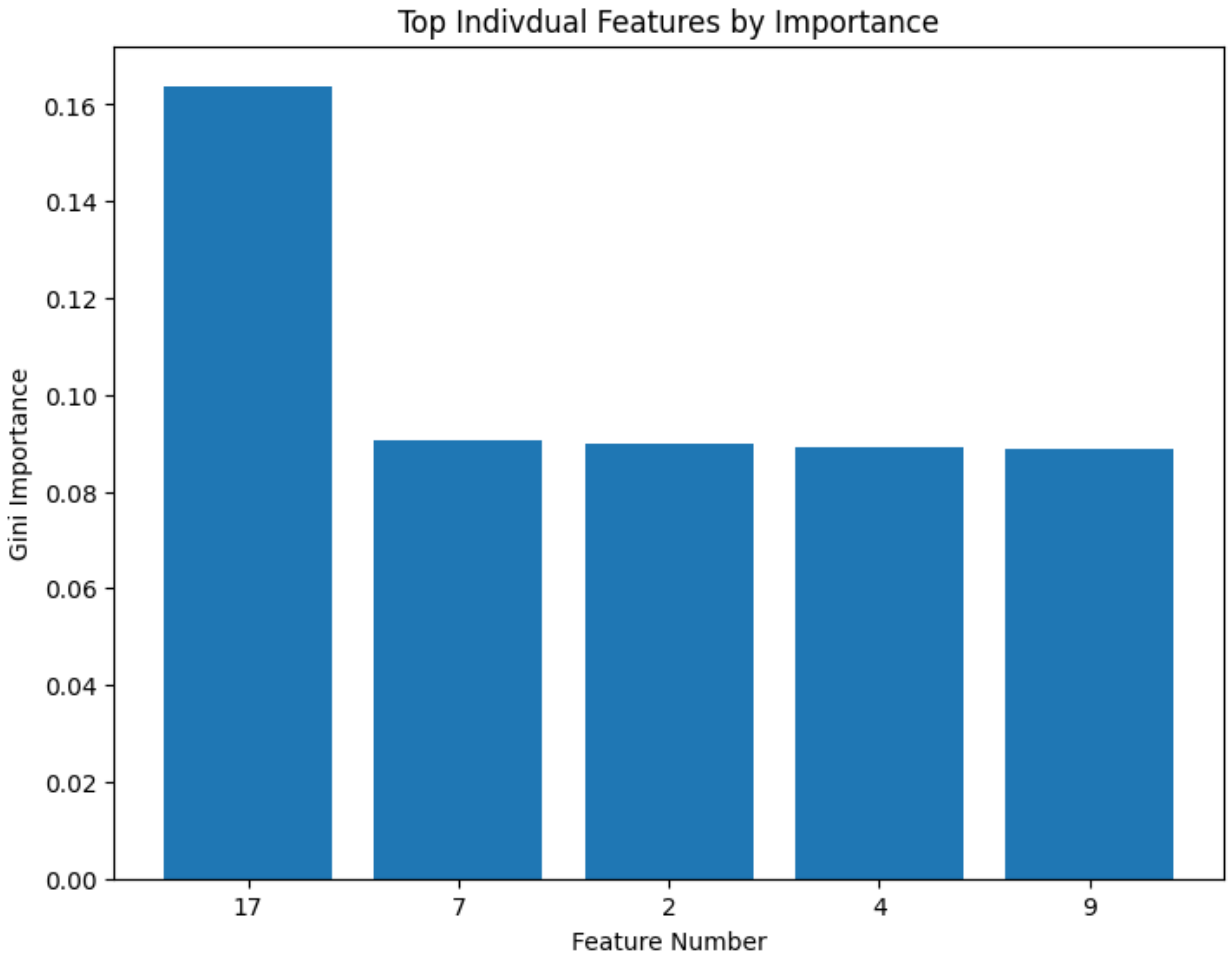
the decision tree rebounded to an accuracy score of 92.45%. The random forest model and k-nearest neighbors model's accuracy scored consistently the same, at 92.46% and 90.08%, respectively. This analysis shows that while these five features certainly assist in training more efficient models, they are not critical to make an accurate diagnosis. Still, the question remains: are there more important features that might affect performance to a greater degree? Using scikit-learn's feature importance function for decision trees, it was possible to calculate the Gini Importance for each feature used in the model. Table 40 shows the top five individual features by importance of the initial decision tree model. Figure 51 shows the importance value for each of those features. The feature

Table 40: Features Removed from Optimal Decision Tree (Gini Importance)

Feature	Feature Number
Containment Temperature	17
Hot Leg 2 Temperature	7
RCS LVL LOOP 1 WR	2
Hot Leg 1 Temperature	4
Cold Leg 2B Temperature	9

with the highest Gini importance in the initial decision tree model was the containment temperature. When removed from the dataset and a new decision tree model trained, the accuracy did decrease to 91.81% from the 92.33%. The random forest and k-nearest neighbor models experienced larger drops from the initial accuracy. The random forest model's accuracy score decreased to 90.27% and the k-nearest neighbors model dropped to 90.37%. Although the decrease was larger with the containment temperature, TPOT was still able to produce an accurate model. The feature with the second highest importance was the second hot leg temperature. The decision tree model trained without the top two features scored an accuracy of 90.38%, only slightly lower than the previous model. The random forest model stayed consistent with a score of 90.28%. This is very similar to the last model trained. The k-nearest neighbor model's accuracy dropped to 89.85%. The decision tree model's accuracy remained consistent when the third feature,

Figure 51: Top 5 Features by Gini Importance



the water level for the wide range of the RCS loop, was dropped, scoring a 90.38% in accuracy. The random forest model remained consistent at 90.28%, while the k-nearest neighbor model scored slightly lower at 89.48%. The fourth feature dropped was the first hot leg temperature. Two of the models experienced improved scores when dropping this feature. The decision tree model scored a 91.8% and the k-nearest neighbors models scored a 90.07% for accuracy. The random forest remained consistent at 90.26%. The final feature dropped was the 2B cold leg temperature. The decision tree model only scored slightly lower with an accuracy of 91.66%. The random forest model scored a 90.29%, a similar score to the others trained using this method. Finally, the k-nearest

neighbors dropped again down to 89.40%. Table 41 shows the validation results for all the models trained for the decision tree analysis. Table 42 shows the difference in validation scores from the base model trained.

Table 41: Validation Results from Feature Removal Analysis)

Model	Accuracy	Precision	Recall	F1 Score
DT W/O Top G. Impurity Feature	92.33	92.09	92.12	92.10
DT W/O Top 2 G. Impurity Features	92.4	92.16	92.19	92.10
DT W/O Top 3 G. Impurity Feature	92.42	92.18	92.19	92.19
DT W/O Top 4 G. Impurity Features	92.36	92.12	92.15	92.13
DT W/O Top 5 G. Impurity Features	92.45	92.22	92.24	92.23
kNN W/O Top G. Impurity Feature	91.6	91.37	91.41	91.36
kNW/O Top 2 G. Impurity Features	92.42	92.42	92.36	92.17
kNN W/O Top 3 G. Impurity Features	91.76	91.52	91.59	91.53
kNN W/O Top 4 G. Impurity Features	91.07	90.85	90.94	90.82
kNN W/O Top 5 G. Impurity Features	91.08	90.85	90.95	90.8
RF W/O Top G. Impurity Feature	92.46	92.24	92.25	92.25
RF W/O Top 2 G. Impurity Features	92.42	92.21	92.22	92.21
RF W/O Top 3 G. Impurity Features	92.30	92.17	92.19	92.18
RF W/O Top 4 G. Impurity Features	92.46	92.25	92.26	92.25
RF W/O Top 5 G. Impurity Features	92.45	92.24	92.24	92.24
DT W/O Top G. Importance Feature	91.81	91.63	91.68	91.58
DT W/O Top 2 G. Importance Features	90.38	90.15	90.10	90.12
DT W/O Top 3 G. Importance Features	90.37	90.11	90.07	90.00
DT W/O Top 4 G. Importance Features	91.82	91.63	91.72	91.72
DT W/O Top 5 G. Importance Features	91.66	91.47	92.58	91.28
kNN W/O Top G. Importance Feature	90.37	90.17	90.23	90.00
kNN W/O Top 2 G.Importance Features	89.85	89.61	89.63	89.53
kNN W/O Top 3 G.Importance Features	8948	89.25	89.26	89.04
kNN w/o Top 4 G.Importance Features	90.07	89.86	89.85	89.71
kNN w/o Top 5 G.Importance Features	89.40	89.16	89.12	89.02
RF w/o Top G.Importance Feature	90.27	90.03	90.00	90.01
RF w/o Top 2 G.Importance Features	90.28	90.04	90.01	90.02
RF w/o Top 3 G.Importance Features	90.29	90.06	90.04	90.04
RF w/o Top 4 G.Importance Features	90.26	90.02	90.00	90.01
RF w/o Top 5 G.Importance Features	90.29	90.06	90.04	90.04

Table 42: Differences In Validation Results from Feature Removal Analysis)

Model	Accuracy Difference	Precision Difference	Recall Difference	F1 Score Difference
DT W/O Top G. Impurity Feature	-0.05	-0.06	-0.03	0.96
DT W/O Top 2 G. Impurity Features	0.02	0.01	0.04	1.03
DT W/O Top 3 G. Impurity Features	0.04	0.03	0.04	1.05
DT W/O Top 4 G. Impurity Features	-0.0	-0.03	0.00	0.99
DT W/O Top 5 G. Impurity Features	0.0	0.07	0.09	1.32
kNN W/O Top G. Impurity Feature	0.45	0.45	0.44	0.22
kNN W/O Top 2 G. Impurity Feature	1.20	1.50	1.39	1.26
kNN W/O Top 3 G. Impurity Features	0.61	0.6	0.62	0.39
kNN W/O Top 4 G. Impurity Features	-0.08	-0.07	-0.03	-0.09
kNN W/O Top 5 G. Impurity Features	-1.35	-0.07	-1.32	-0.08
RF W/O Top G. Impurity Feature	0.03	0.02	-0.02	0.03
RF W/O Top 2 G. Impurity Features	-0.01	-0.01	-0.05	-0.01
RF W/O Top 3 G. Impurity Features	-0.04	-0.05	-0.08	-0.04
RF W/O Top 4 G. Impurity Features	0.03	0.03	-0.01	0.03
RF W/O Top 5 G. Impurity Features	0.02	0.02	-0.03	0.02
DT W/O Top G. Importance Feature	-0.50	-0.52	-0.47	0.44
DT W/O Top 2 G. Importance Features	-2.00	-2.00	-2.03	-1.02
DT W/O Top 3 G. Importance Features	-2.01	-2.04	-2.08	-1.06
DT W/O Top 4 G. Importance Features	-0.56	-0.52	-0.4	0.58
DT W/O Top 5 G. Importance Features	-0.72	-0.68	0.43	0.14
kNN w/o Top G. Importance Feature	-0.78	-0.75	-0.74	-0.91
kNN w/o Top 2 G.Importance Features	-1.30	-1.31	-1.34	-1.38
kNN w/o Top 3 G.Importance Features	-1.67	-1.67	-1.71	-1.87
kNN w/o Top 4 G.Importance Features	-1.073	-1.06	-1.12	-1.2
kNN w/o Top 5 G.Importance Features	-1.75	-1.76	-1.85	-1.89
RF w/o Top G.Importance Feature	-2.16	-2.19	-2.27	-2.21
RF w/o Top 2 G.Importance Features	-2.15	-2.18	-2.26	-2.2
RF w/o Top 3 G.Importance Features	-2.14	-2.16	-2.23	-2.18
RF w/o Top 4 G.Importance Features	-2.17	-2.2	-2.27	-2.21
RF w/o Top 5 G.Importance Features	-2.14	-2.16	-2.23	-2.18

4.8 Discussion

This phase of the experiment aimed to expand the scope of the previous effort by determining if the non-nerual network models could continue to provide accurate diagnosis

when considering a larger number of different reactor transient events. In this phase, data was once again collected from the GPWR simulator to develop machine learning models using TPOT. This resulted in a dataset consisting of over 110,000 points, with 12 different classes of data.

The validation results from the models trained in this study were mixed when compared to the previous section. The three naive Bayes models and logistic regression models suffered significant drops in validation results. However, the decision tree and k-nearest neighbors models, along with the new random forest model, maintained high validation results, in the low 90's. This demonstrates that these types of machine learning models can effectively distinguish between several different types of transient events. Also, all but one of the seven models were able to perfectly identify a reactor operating normally from one experiencing a transient.

To take the analysis of the models to a great level of detail, the time stamp data was used to identify where in the simulation misclassifications occurred. This analysis showed that for the three higher performing models, a higher number of misclassifications occurred within the first five seconds of the transient simulation. After 30 seconds, the model was better able to classify the transients. Past this point in the simulation, the model only experienced an average of 10 misclassifications in each second of the simulation. This indicates that a highly accurate diagnosis could be provided to operators within 30 seconds of the event occurring using these models.

The three best performing models had struggles correctly identifying the electrical load rejection, feedwater pump trip and turbine trip without SCRAM transients, with accuracies in between 60% and 70%. To improve the model different data splits were used with testing and training the data, however this did not lead to a significant change in validation results. Another attempt to improve the model involved allowing TPOT more time to find an optimal model by increasing the number of pipelines retained in each generation, as well as the number of generations the training was ran for. Unfortunately,

this also did not result in any significant change in performance. Since there were no simple changes to apply to the model to fix the misclassification issues, it was important to try and understand the cause of the issues. To do this, the incorrect testing data was divided into subsets based on which transient the data was classified as. The descriptive statistics were then calculated and examined for each of these six subsets. The values were compared to the descriptive statistics for the data that was correctly classified. It was found that the two steam generator levels had significant differences from the correctly classified data for five of the six subsets. This appears to be part of the reason the data was incorrectly misclassified for these cases. In order to address the concern of key data features missing when a diagnosis is needed, the three higher performing models were trained with features removed. Five features were removed based on Gini Impurity and Gini Importance from the optimal model. TPOT was able to train new models that still had high validation results despite missing key features, based on both categories. These results were also in the low 90's. These kinds of models could be used as a contingency in the event sensors are damage or have failed in the course of a transient event. Finally, analysis was done to see how the three higher performing models are affected by changes in the random state used in model training. These results showed that the random state only had a minimal impact on the models, as each of the models only had a variation around 1% across twenty different random states.

Major findings from this phase of the project were -

- A larger number of transient events can be diagnosed accurately using non-neural network models, such as decision trees and k-nearest neighbors.
- In the event key features are missing, TPOT trained models can still make highly accurate diagnoses.
- The random state used to train the higher performing models has a minimal impact on model validation results.

5 Anomaly Detection

The expanded use of data science, machine learning, etc. in the world today, will likely have a large impact on every day life in the near future. As was mentioned earlier, many people already interact with these systems, most without even realizing it. However, there are a number of issues that will need to be addressed as the use of machine learning increases. One of these issues relates to the ability to determine if there is something wrong with the data being used with a machine learning model. This can be the result of equipment failure, such as a sensor malfunction, or even be part of a security breach, such as a hacker altering the data. One approach to deal with this is the use of a machine learning model for the purpose of identifying anomalies in data. This section will explore the use of machine learning anomaly detection with the data used in the reactor transient identification phase of this project.

5.1 Background

5.1.1 Data Security

In recent times, the fear of cyber related attacks has increased. In March of 2021, the Government Accountability Office (GAO) issued a report that strongly recommended DOE update its cybersecurity strategy to include the electrical grid and distribution[79]. The NRC released requirements for nuclear facilities making use of digital assets in 10 CRF 73.54.[80]. The requirements include: the determination of whether the digital asset could be the target of an attack, which mediums could be used to perform such an attack and how an attack would be mitigated. The specifics are left to the individual licensees. Due to this, any machine learning system used with a nuclear power plant will have to have some sort of assessment performed. In machine learning, there are two possible threats to the integrity of a model. The first occurs when the training data for a model is altered. This results in faulty models that do not truly represent the actual data. This

is referred to as data poisoning. The second threat involves altering training data, so the prediction provided by the model is not accurate.

In practice, the probability of data poisoning occurring is fairly low. Personnel that would be involved in implementing a model at a nuclear plant will be highly vetted per United States law. Also, security measures would be in place that would prevent any outside person from interfering in model training. A more likely scenario would be that the outside attacker would look to interfere with the data from the reactor through the sensors. This possible threat can be addressed by training a model to look for anomalies in the testing data before it is actually tested. Once an anomaly is detected, the system could simply switch to a model that doesn't use the feature where the anomaly was detected. If the anomaly is large enough, it could be used as part of a threat detection system. It should be noted that this type of system shouldn't be limited to looking for cyber threats. An anomaly detector could be used as a way to identify sensors that have malfunctioned.

5.1.2 Auto encoders

Auto encoders are a machine learning approach that has in recent years found several different applications. An auto encoder is a type of ANN that doesn't rely on target data i.e. a unsupervised approach. In general, an auto encoder's purpose it to take inputs that may or may not contain statistical noise and reconstruct those inputs resulting in outputs with little or no distortion[81]. The technique was first proposed in the 1980s, but major interest in the approach didn't occur until the renewed interest in ANNs in the early 2010s.

An auto encoder consists of two major pieces. The first is known as an encoder, which coverts the inputs into a latent representation[1]. The decoder takes the representations produced by the encoder and produces the outputs of the model, commonly refereed to as reconstructions. The key with the auto encoder is that the number of neurons in the output layer of this model cannot exceed the number of inputs,that is, the number

of features of the data. Both the encoder and decoder consist of a number of hidden layers known as bottleneck layers. Bottleneck layers are used between the decoder and encoder output layers to compress the data. In these layers the number of neurons decreases from layer to layer in the decoder and increases back up to the original in the encoder. The idea behind this is that the network will be forced to learn the important characteristics of the data. Auto encoders are otherwise implemented in the same manner as other ANNs. For example, an auto encoder used for image modification will likely take the form of a CNN, consisting of convolutional and pooling layers. Auto encoders also make use of activation functions, loss functions, etc. In addition to anomaly detection, auto encoders can be used for feature reduction, in a similar manner as PCA, and image modification.

5.1.3 TensorFlow

Since autoencoders are neural network type models, it will be necessary to use a python neural network library to construct the model. Today there are a number of neural network packages such as Meta/Facebook's PyTorch and Nvidia's CUDA. For this project the Python package Keras will be used, as it is one of the most established neural network tool kits available at this time. Keras is a built in package of the TensorFlow Library and as such this project will rely on both, TernsorFlow and Keras.

TensorFlow is a free open source library focused around developing machine learning algorithms. Originally developed by Alphabet's Google[82], the package is well known for its ability to support neural network models. Version 2.0 of TensorFlow was released by Google in 2019 and was the first public release of TensorFlow[83], but older version are still available and have support. One of the key advantages of TensorFlow is that the package has support for a variety of programming languages including Python, Java and C++.

TensorFlow has been designed to have the optimal ability to use and manipulate the

tensor data structure. A tensor is a type of data flow graph data structure that is used in many linear algebra applications, a key in neural networks. TensorFlow has also been developed to make use of multi-core CPUs, multi-thread GPUs and Google's custom Tensor Processing Unit. According to TensorFlow, a number of large companies other than Google make use of TensorFlow, such as Coca-Cola, Intel, General Electric Health Care and AirBNB[84].

5.1.4 Keras

Keras is a Python deep learning API designed to interface with TensorFlow. Keras is included in the standard installation of TensorFlow. As of 2022, the library supports Python Versions 3.6-3.9 and can be used with Microsoft Windows, Mac OS and Ubuntu operating systems[85]. The library is promoted as an easy to use tool for the development of neural network machine learning models. According to Keras, organizations including NASA, YouTube and Waymo, make use of the library.

Keras has support for a number of different neural network applications. This includes the training of RNNs for things such as natural language processing and CNNs for photo recognition and analysis. Models can be trained using the Sequential, Functional or Subclass API to best suit the user's needs and preferences. Keras has support for a number of different hyperparameters such as batch size, optimization and activation functions. Keras also supports exporting applications to Java and other web-based platforms, such as Java Script.

5.2 Literature Review

Auto encoders have been used in a number of recent studies in machine learning. As mentioned, one proposed use for auto encoders is feature engineering. One study from Peking University in China proposed using an auto encoder feature selector to better identify features for use in machine learning models[86]. According to the paper, one

of the advantages of this type of feature selector method is the ability to consider both linear and non-linear information when considering the importance of the feature. This differs from more traditional approaches which can only consider linear information. In this study, the authors used the auto encoder on a variety of different dataset including the MNIST and Isolet datasets and compared the results to those using five other feature selection techniques, such as Loplacian Score and Unsupervised Discriminative Feature Selection. The results from this test were positive, as the auto encoder approach produced better results in all but one of the datasets. In the one set that it was not the best approach, the auto encoder was a close second. The authors hope that this type of feature selection can help produce machine learning models faster, as data sets becomes more and more complex.

In the area of anomaly detection, there has also been a number of studies done in recent times. One study from the University of Pisa in Italy, explored the use of auto encoders to find defects during the manufacturing of goods. This study combined an auto encoder with a deep neural network and a discriminator to attempt to identify defects in the manufacturing process[87]. The initial results were positive when applied to other case studies according to the authors. The hope is that this type of system can be utilized by operators with little or no data science background.

A study done using auto encoders in the area of cyber security was performed at Yildiz Technical University in Turkey. The authors proposed using auto encoder based models to better detect network attacks on day zero of the attack. The authors note that in many intrusion detection systems there is a high reliance on data from previous attacks. This creates a potential hazard for newer networks, as the system may not have the necessary data needed to detect the breach[88].

To this end more modern network breach detection systems have begun to explore the use of machine learning based systems to detect issues earlier. In this study the author proposed designing a model that would detect anomalies to help indicate the start

of an attack. This approach requires either an unsupervised or semi supervised model, which makes the auto encoder an appealing choice.

This study explored the use of two different types of auto encoders. The first is a traditional auto encoder similar to those described in previous sections and the other was a de-nosing autoencoder, which can be used to explore statistical relationships between data. One of the keys in this study was determining the threshold. This is the amount of reconstruction error that would be lead to a point to be considered an anomaly. Most autoencoders use a deterministic approach to calculate this point. This study used a stochastic approach instead.

The autoencoder models were scored based on accuracy, precision, recall and f1-score. All of these measurements came back positive with results between the upper 80%s and 90%s. These results compared very well with other similar efforts that had been preformed prior. The authors hope that the results from this study will allow for earlier detection of network intrusions.

5.3 Methods

5.3.1 Data Exploration and Preprocessing

Due to better compatibility with TensorFlow and Keras, this portion of the experiment was done using Google's Colaboratory Notebook and cloud resources, rather than INL's HPC. The first step in performing this study was to prepare the data collected from the GPWR simulator to be trained. As was the case in the previous part of the project, a number of features needed to be dropped from the dataset, as they were simply percentages of other features collected and are considered redundant. The same eight features that were dropped in earlier phases of this project, were once again dropped. The descriptive statistics and the shape of the new Pandas dataframe were verified to ensure the process was done correctly. Dummy variables were once again added for reactor core life.

Autoencoders are a form of unsupervised machine learning, as such the target data from the dataset was dropped.Since no AutoML packages were used in this part of the experiment, it was necessary to pre-process the data manually. As mentioned earlier, one of the common techniques for this is standard scaling which scales based on the variance of the data. Due to these factors, it was decided to go with this technique. Normalization would have been another choice for this. This was done using scikit-learn's Standard Scaler method. It should be noted that the features on time and reactor core life were not scaled. This was done because simulation time will not be used in the model training and reactor core life represents a categorical feature.

The next step in preparing the data was to split the data and set data aside for testing and training purposes. As was done previously, this was done with scikit learn's test _train_split method, with half the data used for training and half for testing. In order for the autoencoder's ability to detect anomalies to be tested, it was necessary to alter a portion of the data with random statistical noise. First, a quarter of the testing data, approximately 13,884 points, were split off from the dataset. This data was modified by adding random noise with an average of zero, to better represent the scaled data, and a standard deviation of two. The noise was created using Numpy's random function. The size of the modified datafame was verified and the dummy variables added back to the dataframe. A column was created to indicate both, the clean and noisy data, to indicate if the data was unaltered or altered. Clean data was assigned a 1 and noisy data a 0. This will be used in validation later on and dropped before testing. A dataframe was created containing both, clean and noisy data. All data used in training will consist of the 27 features used in both earlier studies.

5.3.2 Building the Autoencoder

The autoencdoer built for this experiment was trained using Keras. The architecture for this model included an input layer, an encoder and a decoder. The input layer used a

defined shape of 27 for the features to be trained on. The encoder consisting of four hidden layers, three dense layers and a single dropout layer, with a dropout rate of 0.2. The Relu activation function was used in all of these layers. The number of neurons used in the dense layers halved with each layer, starting at 27 and ending at 7. The decoder also consisted of four hidden layers. The dropout layer was the same as that in the encoder. The first dense layer consisted of 14 neurons and the remaining two used 27. Again the Relu activation function was used. The summary of the model was printed and can be seen in Figure 52.

Figure 52: Keras Summary of Autoencoder

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 27)]	0
dense (Dense)	(None, 27)	756
dropout (Dropout)	(None, 27)	0
dense_1 (Dense)	(None, 14)	392
dense_2 (Dense)	(None, 7)	105
dense_3 (Dense)	(None, 14)	112
dropout_1 (Dropout)	(None, 14)	0
dense_4 (Dense)	(None, 27)	405
dense_5 (Dense)	(None, 27)	756

Total params: 2,526
Trainable params: 2,526
Non-trainable params: 0

5.3.3 Training the Autoencoder

With the parameters of the autoencoder defined, it now was possible to compile and train the model. The Adam optimizer was used for compiling the model and loss was calculated using Mean Absolute Error. The model was then fit using only the data set aside for training, approximately 55,000 data points. The batch size was set to 100. 100 epochs were ran for training. Validation was done using only the unaltered portion of testing data. The training process took approximately 160 seconds. The training loss and validation loss of each epoch was recorded and graphed for comparison. This can be seen in Figure 53.

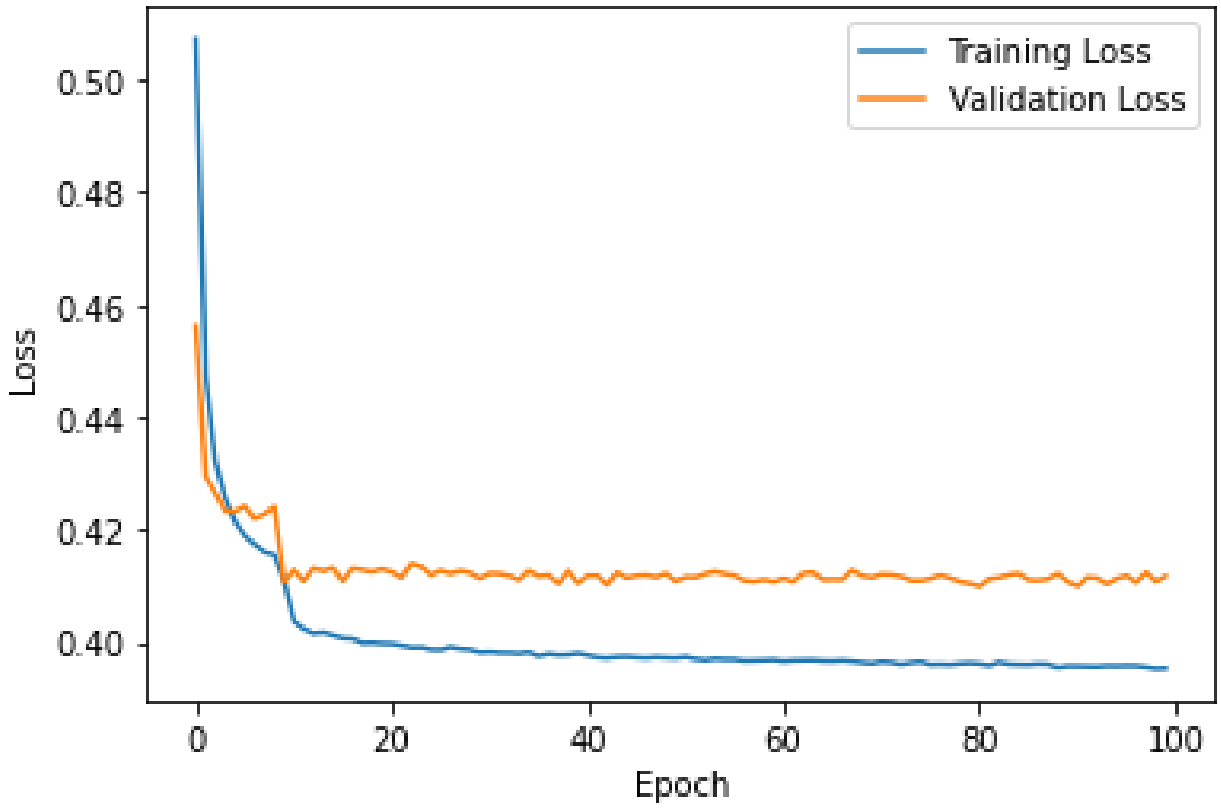


Figure 53: Training Vs. Validation Loss of Autoencoder

5.3.4 Validating the Autoencoder

The first step in determining how the Autoencoder performed is to determine the reconstruction error from both, the clean and noisy data. To do this, it was necessary to use the Autoencoder to predict both, the noisy and clean data. Once that was done, it was possible to begin calculating the reconstruction error using NumPy. In this case, the Mean Squared Error was calculated for each point in both, the clean and noisy datasets. Once this was determined, a threshold was identified to determine if a point is an anomaly or not. There are a number of ways this can be done, for example, this can be done visually by graphing the reconstruction error of the points or can be done mathematically. In this case, the threshold was set to average training loss of the model plus a single standard deviation of the training loss, yielding a threshold reconstruction error of approximately 1.83.

Once the threshold is established, it is possible to determine how many points are considered to be anomalies in the dataset. Any point that contained a reconstruction error above 1.83 is consider an anomaly. From this the number of True positives, false positives and false negatives can be determined. This data was used to calculate the accuracy, precision, recall and f1-score for the model. This process was repeated for three lower levels of noise to determine how the autoencoder performs when the altered data is modified to a smaller degree.

5.4 Results

The results from the initial test of the autoencoder were positive. Of the over 55,000 samples tested with the autoencoder, the model scored an 91.05% in accuracy, or 55,561 of the data points tested points correctly. At this threshold and level of noise, the model was able to nearly perfectly distinguish the altered points correctly. The model only identified 13 of the over 13,000 altered data points as unaltered data for a recall of 99.90%. The

model did have some issues with a group of data from the set. This resulted in the model identifying 4959 of the over 41,000 unaltered points as anomalies. This gives the model a scored precision of only 73.66%. The resulting f1-score for this model was 84.80%. The confusion matrix for this model can be seen in Figure 54.

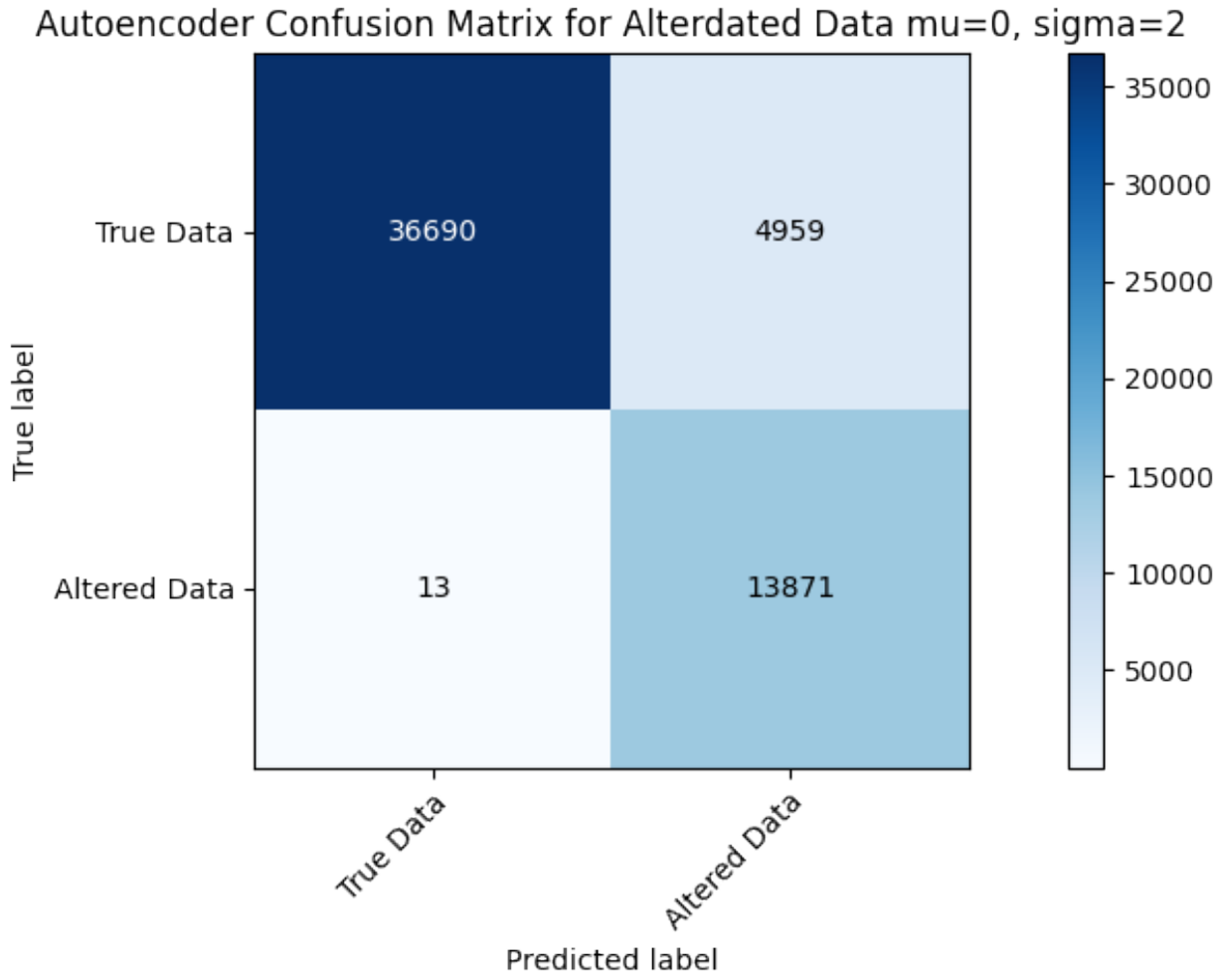


Figure 54: Confusion Matrix For Autoencoder Results

Through visual inspection, it can be seen that all of the 6379 false positive points had a reconstruction error between 1.83 and 6.00. This is almost certain the result of portions of the dataset being considerably different than others. This is not unexpected as this dataset contains data on 12 different reactor events and it is expected that some of this data will significantly vary from transient to transient, especially in different points

in time during the transient. A graph of the reconstruction error of the clean data points can be seen in Figure 55.In terms of altered data points, the vast majority of altered data points, over 11,000 points, had a reconstruction error between 1.83 and 6. The remaining points, approximately modified 2000 points, had reconstruction error between 8 and 14 and were easily identified by the model. Figure 56 shows the plot of the altered data's reconstruction error. Figure 57 shows the plot of both, the altered and clean data's reconstruction error.

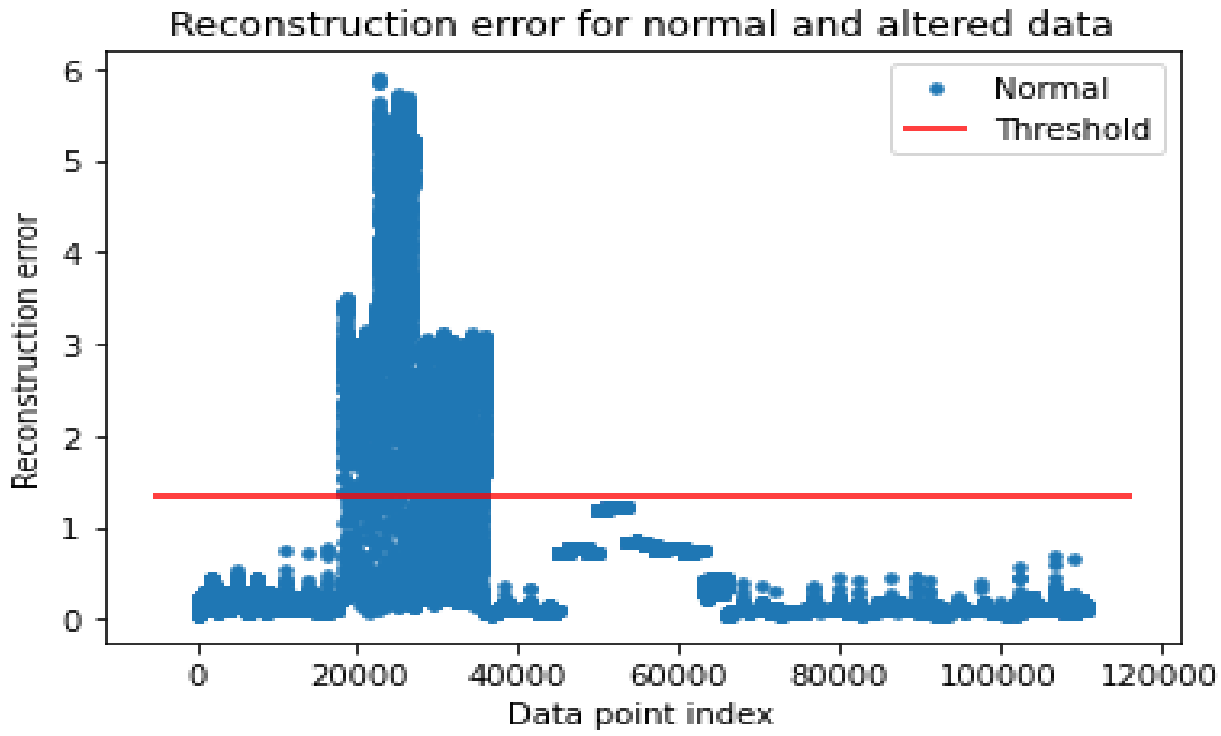


Figure 55: Reconstruction Error for Clean Data Points

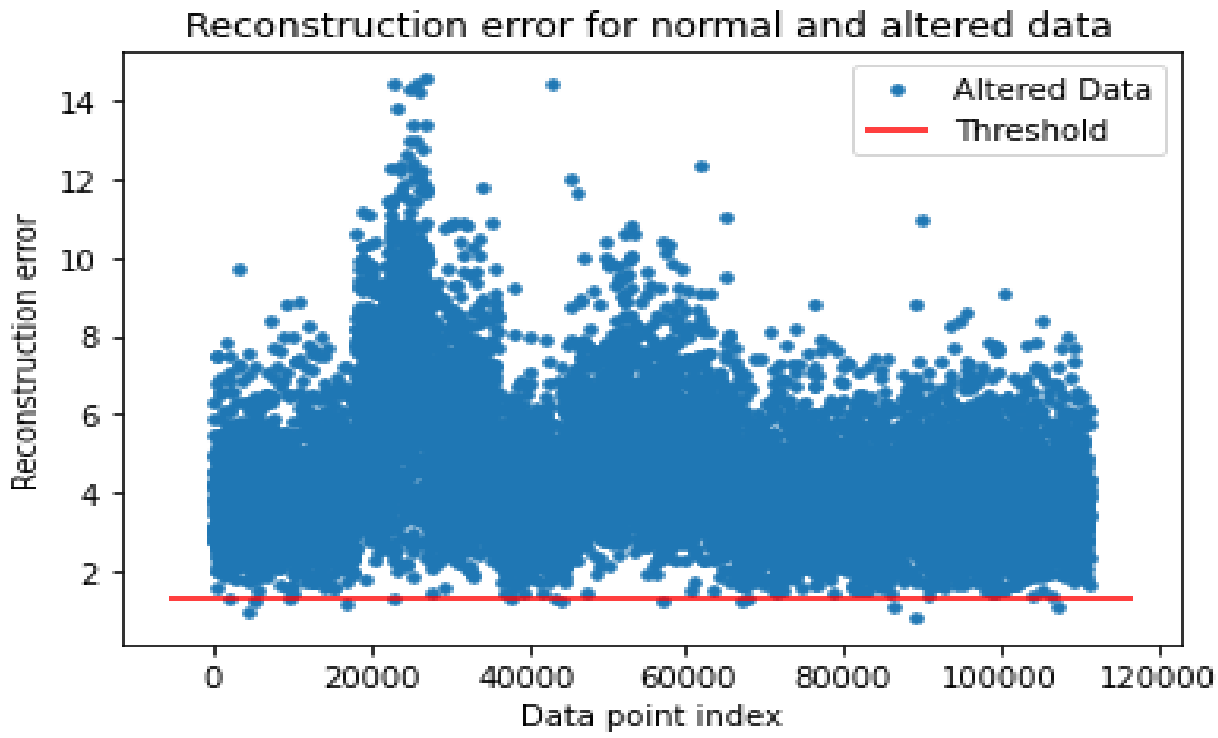


Figure 56: Reconstruction Error for Altered Data Points

Reducing the amount of noise used in altering the data had a adverse effect on the model. As less noise was introduced into the data, the autoencoder had a more difficult time identifying the altered points,as expected. When the noise was reduced from two standard deviations from the mean to just 1.5, the impact was fairly minimal. The threshold was reduced slightly to 1.13. Despite the lower amount of noise and the reduced threshold, the autoencoder was still able to correctly identify the majority of the points. Over 48,000 were correctly identified as anomalies or normal data with this level of noise. The accuracy of this model was calculated to be 87.94%, a less than 4% difference when tested on noisier data.

The testing of this autoencoder on this data showed a continued trend with the model in terms of precision and recall. Once again, the autoencoder did a good job in classifying altered data correctly, with only 200 altered points being classified as normal. This gave the model a recall of 98.56%, very close to the recall when tested on the noisier data. Also,

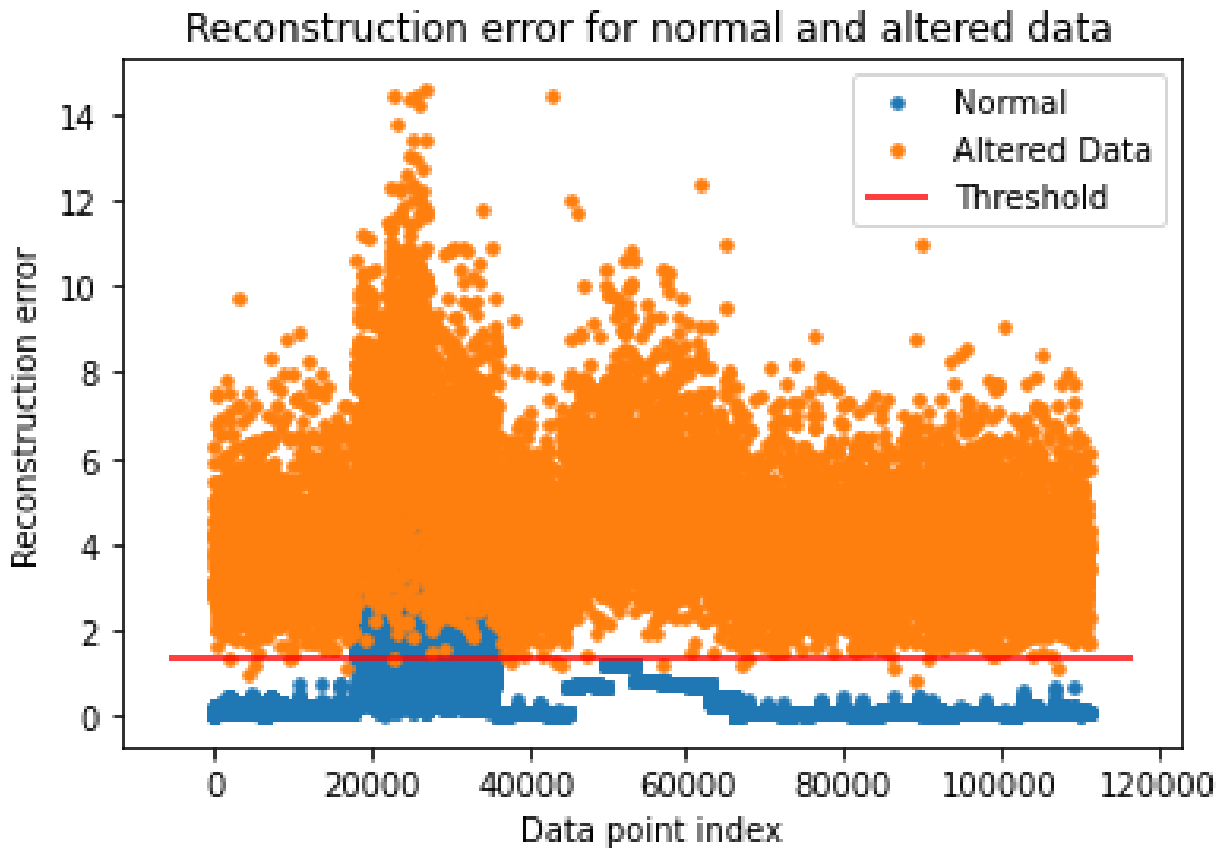


Figure 57: Reconstruction Error for All Data Points

the model had close to the same difficulty correctly distinguishing normal data from altered data, with 6496 normal points being incorrectly classified as altered. This yielded a precision of 67.8%.The f1-score from this testing was 80.34%.

The pattern of decreasing performance as noise decreased continued as the noise in the data was decreased to 1 standard deviation and again, to 0.5 standard deviation from the mean. The accuracy when tested with the single standard deviation data, was significantly lower, at only 77.8%. When lowered again to 0.5, the model accuracy was once again significantly lower at only 66.7%. It was at a single standard deviation where the recall of the model began to suffer. The model incorrectly identified noisy data as normal, with over 3500 noisy points incorrectly identified, resulting in a recall of 74.38%. When the model was tested at 0.5 standard deviation, the recall dropped down to 33.5%.

This result was not unexpected, as the reduced noise and threshold would result in much lower reconstruction error.

In terms of precision, altering the threshold for the data points also caused the results to decrease. At a single standard deviation, the precision dropped to 54.07% and down to 33.57% at 0.5 standard deviation. Again, this was not unexpected, however it should be noted had the threshold been left at the original 1.83, the score would have remained the same as in the previous test. However, the recall would likely have also decreased. The f1-score for the single and half standard deviation test were 66.62% and 33.57%, respectively. Table 43 summarizes the results for all four tests done with the autoencoder. The confusion matrices and reconstruction error plots for the 1.5,1 and 0.5 standard deviation test can be seen in the Appendix section.

Table 43: Results From Autoencoder Test

Level of Noise	Accuracy	Precision	Recall	F1 Score
Mean=0, Standard Deviation=2	91.05%	73.66%	99.90%	84.80%
Mean=0, Standard Deviation=1.5	87.94%	67.8%	98.56%	80.34%
Mean=0, Standard Deviation=1	77.80%	54.07%	74.38%	62.62%
Mean=0, Standard Deviation=0.5	66.72%	33.50%	33.58%	33.54%

5.5 Discussion

These results show that autoencoder based models do have the ability to detect data points with high levels of noise within the dataset collected from the GPWR simulator. The high validation marks with the data altered by two standard deviations from the mean of the standard scaled data are encouraging. This autoencoder was able to produce results above 90% for this level of noise. Another encouraging result from this study is that the autoencoder can easily distinguish the normal points from those with the higher level of noise, with a recall in the high 90% range for both, high noise data points. It should be noted that this model leans towards prioritizing the identification of false

negatives, rather than false positives. This is shown with the lower precision results of the model when tested with the noisy data.

The lower precision of the autoencoder model at higher levels of noise does imply policies will need to be implemented regarding when to take action and when investigation is needed. In the case of the autoecnoder trained for this model, there were three distinct regions of the error reconstruction plot. The first was between 0 and the threshold. In the higher level noise test, the points in the region were nearly all normal points. Points found to be in this region have a high confidence that they are safe for use. The second region is the area from the threshold to about 6.0. Here, the majority of the points are altered and the model classified them as such, however, there are a notable number of normal data points in the region. It would be prudent to investigate and possibly take preliminary action when points are in this region. Finally, the third region exists from 6.0 and higher. Points in this region, while fewer than those in the previous area, are exclusively noisy points. These points should be disregarded right away and the cause of the issue investigated.

The autoencoder did struggle as the noise level decreased in the altered data. At 0.5 standard deviations from mean the model was unable to effectively distinguish the noisy points from the normal points. While this is a concern, it should be noted that the lower noise in the dataset, the less likely the noise will adversely affect the classifying model. When the optimal decision tree was tested with the a low level of noise, 0.1 standard deviation from the mean, the model still maintained its accuracy.

It should also be noted that autoencoders and other anomaly detector approaches should only be a small piece of any cyber security system with a machine learning based safety system. Ideally, systems will be in place to prevent any type of breach before the autoencoder is needed. Prevention and proactive security is a more effective approach than implementing reactive measures. Any facility looking to apply machine learning, or any digital based system for reactor monitoring, needs to implement robust security

measures. This includes administration controls and policies that encourage best safety practices, as well as physical/cyber systems designed to only allow authorized individuals access to sensitive parts of a reactor system. The NRC and DOE have a history of implementing and mandating high security measures at nuclear facilities. These organizations will need to continue to be proactive in their approach to security.

One final note, is that the use of autoencoders should not be limited to just security applications. As the models are able to detect high levels of noise, autoencoders could also be used in the area of detecting failures with equipment. Nuclear power plants rely on a number of sensors and instruments to provide operators at the facility information needed to properly run the facility. The use of an autoencoder to detect malfunctions and failures with these type of sensors would be an approach worth exploring. For example, if a sensor were to fail, the autoencoder should detect the noise from the data being transmitted to the operator. This could potentially allows for quicker identifications of equipment failures, such as the PORV failure at TMI. This would add another layer of protection to nuclear power and contribute to the safety efforts at power plants.

6 AutoML Comparison

The use of machine learning and artificial intelligence has grown significantly in research and business as shown in the Background Chapter. Large companies such as Coca-Cola, Spotify, Amazon and G.E. has invested large amount of resources into machine learning programs. However, it should be noted that these companies have vast resources and can have dedicated departments and teams to develop models for use. In order for machine learning to become a standard in industry, it will be necessary for smaller and smaller companies to adopt the technology. One study showed that in Germany about 25% of companies make use of machine learning and artificial intelligence in operations[89]. The same study noted that only 10% of smaller and medium sized companies have implemented machine learning. There are a number of potential reasons for this. Smaller companies tend to have fewer staff involved in information science and that staff generally has minimal experience dealing with machine learning. Also, with more limited resources, companies may be reluctant to invest a great deal of resources in an area the majority have significant little understanding in.

To address this gap in the implementation of machine learning and predictive analytics there are a couple of solutions. The first is the development of dedicated firms who can perform the work for the small businesses or companies. There is great potential in this, as small firms maybe use to outsourcing similar work, such as accounting, marketing, etc. Companies such as Google already offer many of these services. This does have the drawback that firms would have to share their information with outside groups and there could be reluctance to do this. A company's information can contain things like trade secrets, client information, etc. This likely also pertains to a company's competitive advantage and is highly sensitive.

The second approach to deal with gap is to simplify the process of training machine learning models. Simplifying the process could allow for fewer dedicated staff in performing the analysis, reduce costs and reduce the level of expertise needed to perform

the work. All of these could lead to higher implementation by firms. As has been noted in this study, AutoML is one approach that can be used to simplify the process of training machine learning models. This section of this effort seeks to examine and compare three different AutoML packages. These are TPOT, H2O AutoML and Google's Cloud AutoML service. The factors that will be compared are performance, time, functionality and ease of use.

6.1 Background

The use and history of TPOT was discussed in a previous section in this report. Due to this, this section will focus on the H2O and Google's Auto ML Service.

6.1.1 H2O AutoML

H2O AutoML is a free open source software package that began its development in 2012.The software is under development by the independent company H2O.ai. The founder and original developer of the package was Sri Ambati, who continues to run the organization as of 2022[90].

H2O was developed to simplify the machine learning process and allow users of all levels of expertise the ability to train models. The organization claims that the software can be used at the enterprise level. The software was developed in Java and has several APIs that allow for use in several different languages, including R and Python and a Graphical User Interface (GUI) called flow is in the early stages of deployment[91]. The first official release of H2O occurred in 2017 though other versions were available as far back as 2013.H2O supports the training of regression, binary and multi-class machine learning models using tabular datasets. Similar to other AutoML packages, H2O begins work in the preprocessing stage of model training. Data exploration and wrangling are still left up to the user. Unlike TPOT however, H2O has included functionality for categorical data. H2O.ai is also looking to develop better algorithm support for feature reduc-

tion and selection techniques. It should be noted that all functionality in H2O is based on H2O.ai's own developed algorithms, not those implemented by another organization, such as scikit-learn. Though there is support for use of basic scikit-learn functions, such as .fit() and .predict(). In terms of models, H2O has a variety of different techniques developed for the AutoML package. This includes random forest classification and regression, support for XGBoost, as well as deep neural networks. The package does make use of cross validation and has support for GPU accelerated training. One important factor to make note of is hyperparameter tuning. H2O makes use of benchmark to determine the ranges for tuning, rather than allowing the user to specify the range. The benchmarks are continuously evaluated to ensure quality. Similar to TPOT, H2O will make use of ensembles to train better performing models.

One of the key highlights from H2O's documentation and published review is the idea of reducing input parameters for the user. Typically, the user should only have to specify the feature data, target data and any limits on the number of models to train and the time allowed for training. Another key feature in H2O is the ability to store what is known as an object called a leader board for the models trained, which allows the user to compare the different models trained using the package.

6.1.2 Google Cloud AutoML

In the area of data analytics there are few companies that have the capabilities of Alphabet's Google. Known for its internet search engine, web marketing operations, as well as role in developing TensorFlow, it is only natural that the company offer machine learning as a service. To this end, Google has developed Google Cloud AutoML, a service designed to offer users access to tools needed to create machine learning models with little experience. Google Cloud AutoML is a paid subscription based service offered by Google. The service not only offers assistance training machine learning models, but also allows users the ability to make use of Google's HPC capabilities. The service provides functionality

for users to train models for a variety of specialized applications. This includes natural language processing, text processing, language translations, video analysis, etc[92]. For the purposes of this study, the AutoML Tables service will be utilized.

AutoML Tables allows the user to make use of a specialized GUI to prepare the dataset for training. Data is uploaded to the Google system via the user's cloud storage, computer or repository. Data wrangling and exploration is highly encouraged by Google and the service will provide information to assist the user in this process. For example, the software will inform the user of the number of missing values and null-able data points. Categorical features can be specified as well. From here, the data can be explored by the user and changes can be made to the dataset. The user then specifies the target data and training can begin[93]. AutoML Tables has a number of parameters that can be specified by the user. The most important one is the number of hours training should be run, as the service does charge by the hour for model training. It should be noted that the process does include preprocessing and the user is not charged for those operations. Google does provide guidelines for the amount of time necessary to effectively train models. This is shown in a screenshot from the documentation in Figure 58.

Rows	Suggested training time
Less than 100,000	1-3 hours
100,000 - 1,000,000	1-6 hours
1,000,000 - 10,000,000	1-12 hours
More than 10,000,000	3 - 24 hours

Figure 58: Timing Guidelines for Google's AutoML Tables Model[93]

Although AutoML Tables uses proprietary algorithms, Google does provide some information on the models being trained. Similar to TPOT and H2O, AutoML Tables trains models using ensemble methods, gradient boosting, decision trees, etc. The software also makes use of neural networks, AdaNet and other machine learning techniques. Other

AutoML services, such as BigQuery ML, allow for more user customization. Once the model has been trained, the software will provide testing results from the model using independent testing points. In the area of classification, the standard four measurements of accuracy, precision, recall and f1 score are provided. The software will also automatically create a confusion matrix based on the results. A feature importance chart is also produced. Finally, the user also has the ability to export the final model that was produced by the software. Support is also provided if the user wishes to test new data using the exported model with Google's resources.

6.2 Literature Review

In addition to the two studies done as part of this project, AutoML packages have been used in a number of other research projects. This section will explore some of the different research studies that have been done using TPOT and H2O AutoML packages, as well as the Google AutoML service.

6.2.1 TPOT

One recent study that made great use of TPOT was a collaboration between the University of Pennsylvania and researchers in the United Kingdom, Finland and the Netherlands. In this study, the authors made use of TPOT to help generate models used to diagnose Connery Artery Disease[94]. Data for this study was obtained from the Angiography and Genes Study database. The final dataset consisted of 73 metabolic features and 27 demographic and clinical features. The study made use of the full TPOT classifier with 14 feature selectors and 11 feature processors. The authors chose to evaluate a number of different models including random forest, logistic regression and Bernoulli Naive Bayes. The optimal TPOT model for this study was a Bernoulli Naive Bates model, which made use of 4 different preporcessing methods. This model achieved a balanced accuracy of close to 78%. The authors hope to continue applying this methods to expand

computer aided diagnostics. Another study that made use of TPOT took place in the field on neuroscience. Researchers in the United Kingdom used TPOT to create models that predict the age of a subject's brain[95]. This study made use of TPOT's regression model capabilities. In this study, the authors made use of over 10,000 MRI results for patients between the ages of 18 and 89. The results were obtained from 13 different publicly available medical databases. Training was done for a relatively short time, only 10 generations for about six hours. The authors made use of TPOT to find a model with the lowest mean absolute error. A number of different models were produced using techniques such as random forest regression, kNN regression, SVM regression, etc. Researchers found that many of these models produced results that compared very well to other techniques that had been tried in other works. The authors noted the positive results indicated that TPOT trained models performed well in estimating a subject true brain age.

6.2.2 H2O

H2O AutoML has also been the center of a number of research studies in recent years. One such study, out of India, made use of the package to predict the patterns of infection from the COVID-19 virus[96]. This study made use of publicly available data from a number of sources. This included the Kaggle and the World Health Organization. Over 150,000 data points were used in this study. These consisted of information such as age, gender and clinical manifestation. Much of this data was either categorical or ordinal in nature.

One of the goals from this study was to use this data to predict the recovery of a patient from COVID-19. The results from this experiment yielded four different machine learning models. These were a kNN, decision tree, logistic regression and naive Bayes. All of these models scored in the mid to high 90%s, with the logistic regression scoring the highest in the areas of accuracy, specificity and sensitivity.

Researchers have used AutoML in the training of neural network based models as

well. One example of H2O being used in this area was a study from Portugal, which looked to train a CNN to detect surface defects[97]. As is the case with most CNN models, the authors used image data to train their model. The authors used several well known architectures, including ResNet and DenseNet to create a "CNN Fusion" architecture that combines the weights from each model. The data used in this experiment consisted of over 1000 images, with only 10% containing defects. The results from this experiment were very positive with many of the models scoring near perfect scores.

6.2.3 Google Cloud AutoML

Recently, researchers have begun to make use of the many tools offered by Google's Cloud AutoML service. One of these efforts has taken place in Malaysia, where researchers looked to use these tools in the area of Biomechanics[98]. In this case, the study made use of the AutoML Tables services offered by Google. In this study, data was collected using mobile smartphone applications to capture data on the different motions being performed by the user, such as kicking angle and flex input. 1000 samples were taken with four features and one target.

The purpose of this study was to try and classify a person as either, an athlete or an non-athlete, based on their movements. The models trained by AutoML Tables produced near perfect results and identified two features as key in the classification. The researchers hope to expand the scope of this research and make continued use of Google's AutoML services.

6.3 Methodology

The purpose of this part of the project was to compare the three AutoML packages to one another. This was to be done using only the default settings of each package with as little customization as possible. This will allows for a better analysis of ease of use and functionality.

In testing TPOT, no custom dictionary was used, instead the TPOTClassifier was configured to run for 100 generations with a population size of 100, just as was done in previous parts of this project. As was the case in the earlier stages of the project, the data needed to be explored and configured for training using Pandas. Target data had to be assigned a numeric value and dummy variables had to be configured. Data had to be split using scikit-learn. Once the ideal model was trained and scored, the optimal model was exported and a confusion matrix created using scikit-learn.

H2O is a very different package from TPOT. Instead of relying on Pandas and scikit-learn, H2O has been designed as an all in one tool. All functionality in H2O is only compatible with the H2O data structure, the H2O Dataframe. Although H2O has GUI support through Flow, it was decided to use the Python H2O package. For simplicity, the GPWR was imported using Pandas, and the dataset reduced to the same 26 features as the dataset used in the TPOT training. This could have been done with H2O, if needed. It is important to note, the data was not otherwise altered, no dummy variables were configured and no changes to target data were done. The Pandas dataframe was then converted to an H2O Dataframe.

Once the data was in the correct format, it was possible to begin using H2O tools to explore the dataset. The H2O dataframe has similar tools to those found in Pandas and the .describe() method was used to ensure data was ready for training. The data was split in half using H2O's split_frame method. The H2O AutoML instance was configured with a maximum number of models set at 20, max run time of 10,000 seconds and a scoring metric of accuracy. This can be seen in Figure 59. Due to an issue with INL HPC not having a Python kernel with H2O installed at the time, it was necessary to use Google's Colaboratory notebook and server to train the model. As there is a limit to the amount of memory this system allows, the run time was only allowed for three hours.

Once the H2O model was configured, it was possible to begin training the model. H2O required the range of the feature data and the column with the target data be specified,

```
[ ]  #H2o Parameters
     from h2o.automl import H2OAutoML
     aml = H2OAutoML(max_models=20, max_runtime_secs = 10000, seed=1, sort_metric = "accuracy")
```

Figure 59: H2O AutoML Configuration

along with the frame to be used. H2O ran for the allotted time and produced validation results. It is important to note that the H2O AutoML instance is used for both, regression and classification models and the package makes the determination on which model to use, based on the target data provided. In addition to producing the validation results from the training, H2O also provided a table of listing the feature importance for the best producing model. A H2O leader board was also produced showing the top models produced during the training. The model with the highest accuracy was saved and used to produce a confusion matrix using H2O's functionality.

Google's AutoML Tables is also a very different approach than both, TPOT and H2O AutoML. Google's AutoML Tables uses only a GUI, over a web browser, rather than coding, as it was in the previous two approaches. AutoML Tables functions similarly to a Windows Installation Wizard, where the software will guide the user through the major steps. The first of the these is importing the data. Data was uploaded to the AutoML Tables Console directly from the computer. This must be in a .CSV format.

The next step in the AutoML Tables process was to explore the data and train the model. The console does provide support with data exploration, providing a quick analysis of all fields in the imported dataset. Information provided included data type of the field, number of missing values and distinct values. As was the case with the other two tests, the dataset was reduced to 26 features. The default split of 80-20 was used, as Google AutoML Tables requires that the user modifies the .CVS file to change the split. The target field was specified and the AutoML program run for three hours, as directed by Google's guidelines.

Once the AutoML program finished running, the AutoML Tables console provided an

overview of validation results from the optimal model produced. This included area under the curve for both, the precision-recall and receiver operating characteristic curves, as well as precision, recall, f1 score and log-loss. Google AutoML also provided information on feature importance, as well as the balance between the different classes used in the training. Typically, the console will provide a confusion matrix, but since the number of classes was higher than 10, it was unable to do this.

6.4 Results

Although the TPOT model ran for the full 100 generations, over 20 hours, it was possible to track the model's performance over time due to the enhanced verbosity that shows the time of each generation's completion. This is enabled when using TPOT's default settings. Since the other two approaches were only ran for three hours, the optimal model produced at this time was examined, around 17 generations. This model was a XGBoost model with an estimated validation accuracy of 93.32%. Further examination of this model showed the model was able to better predict the turbine trip without SCRAM that those trained in the previous part of the project, over 90%. However, the model had to sacrifice accuracy with the electrical load rejection transient, scored below 60%, lower than those from the optimal models in the previous section.

The H2O AutoML test was able to produce a number of highly accurate model within the three hour time frame allotted. In total, ten models were trained during the run. The leader board from this run can be seen in Figure 61. The model with the highest accuracy was a stacked ensemble method. This model scored a 93.77% accuracy. Other high performing models made use of XGBoost, as did TPOT. Examination of the confusion matrix produced from this optimal model shows that similarly as in the case with the TPOT models, the H2O model struggled with the Feedwater Pump Trip, Electrical Load Rejection and Turbine trip without SCRAM model. The confusion matrix for this model is shown in Figure 60.

	TRANSIENT- Normal Ops	Transient Load Rejection	Transient Rapid Power Change	Transient Single Coolant Pump Trip	Transient Total Coolant Pump Trip	Transient Turbine Trip No SCRAM	Transient Valve Closure	Transient- Depressurization	Transient- Max Steam Line Rupture	Transient- Feedwater Pump Trip	Transient- LOCA LOOP	Transient- Manual Trip	Error	Rate
0	8940.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0 / 8,940
1	0.0	2811.0	0.0	0.0	0.0	762.0	0.0	0.0	0.0	920.0	0.0	0.0	0.374380	1,682 / 4,493
2	0.0	0.0	1525.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.000655	1 / 1,526
3	0.0	3.0	0.0	4480.0	1.0	7.0	5.0	0.0	0.0	4.0	0.0	4.0	0.005329	24 / 4,504
4	0.0	9.0	1.0	2.0	4411.0	8.0	3.0	0.0	0.0	7.0	0.0	3.0	0.007426	33 / 4,444
5	0.0	661.0	0.0	0.0	0.0	3470.0	0.0	0.0	0.0	355.0	0.0	6.0	0.227516	1,022 / 4,492
6	0.0	13.0	2.0	7.0	3.0	8.0	4418.0	0.0	0.0	3.0	0.0	0.0	0.008083	36 / 4,454
7	0.0	0.0	2.0	2.0	0.0	1.0	2.0	4561.0	0.0	1.0	0.0	0.0	0.001751	8 / 4,569
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4553.0	0.0	0.0	0.0	0.000000	0 / 4,553
9	0.0	851.0	0.0	1.0	0.0	354.0	2.0	0.0	0.0	3274.0	0.0	0.0	0.269523	1,208 / 4,482
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4539.0	0.0	0.000000	0 / 4,539
11	0.0	16.0	0.0	0.0	0.0	21.0	0.0	0.0	0.0	8.0	0.0	4476.0	0.009954	45 / 4,521
12	8940.0	4364.0	1530.0	4492.0	4415.0	4631.0	4431.0	4561.0	4553.0	4572.0	4539.0	4489.0	0.073113	4,069 / 55,517

Figure 60: H2O AutoML Confusion Matrix

model_id	accuracy	mean_per_class_error	logloss	rmse	mse
DeepLearning_1_AutoML_2_20220213_10016	0.561234	0.450289	5.02348	0.653244	0.426727
GLM_1_AutoML_2_20220213_10016	0.764856	0.241596	0.577845	0.464297	0.215572
StackedEnsemble_BestOfFamily_3_AutoML_2_20220213_10016	0.873193	0.130132	0.397183	0.364447	0.132821
GBM_5_AutoML_2_20220213_10016	0.897784	0.104943	0.495864	0.395582	0.156485
XGBoost_3_AutoML_2_20220213_10016	0.909287	0.0931476	0.249569	0.289767	0.0839651
XRT_1_AutoML_2_20220213_10016	0.934976	0.0668048	0.168675	0.228823	0.05236
GBM_2_AutoML_2_20220213_10016	0.935912	0.065864	0.138777	0.224878	0.0505701
XGBoost_2_AutoML_2_20220213_10016	0.937281	0.0644055	0.115058	0.202758	0.0411107
XGBoost_1_AutoML_2_20220213_10016	0.937479	0.0642049	0.117816	0.204857	0.0419666
StackedEnsemble_BestOfFamily_2_AutoML_2_20220213_10016	0.937785	0.0639576	0.114576	0.203984	0.0416093

Figure 61: H2O AutoML Leaderboard

The validation results provided by Google's AutoML Tables is unique in that no measurement for accuracy was given. However, the precision result was very high at 97%, higher than the results from the TPOT and H2O runs. That said, recall was in the same range as the other models, at 91.5%. This resulted in a f1 score of 94.1%. As was mentioned earlier, no confusion matrix was provided due to the number of classes used in training. This makes it difficult to provide an accuracy number. However, AutoML Tables does provide a log-loss value, the cross entropy value between the predictions and actual values provided. According to Google, the closer the log-loss value is to 0, the higher quality the model is. The log-loss for this model was 0.105. Figure 62 shows the results, as provided by AutoML Tables. It should be noted, that AutoML Tables does not provide the user with the type of model that was produced or any summary of what techniques were used to produce the model.

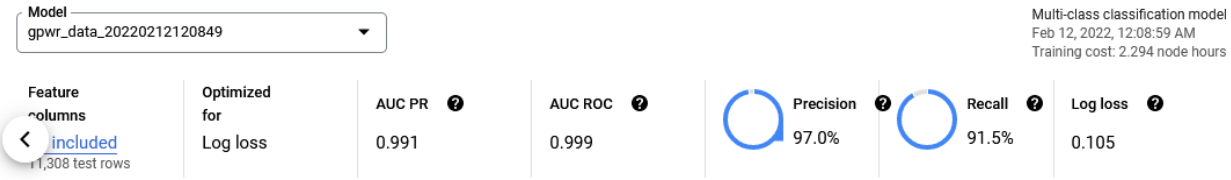


Figure 62: Google AutoML Tables Output

6.5 Discussion

6.5.1 Performance

TPOT, H2O AutoML and AutoML Tables were able to produce high performance models in a short time through the use of HPC. Both, TPOT and H2O AutoML, were able to produce models with accuracies over 93%, although both models struggled with the same three transient events. Although there was no accuracy measure for the AutoML Tables model, it can be inferred through the high precision and recall measurements that the models scored very high, certainly higher than the other two tests. The precision recall and f1 score from both, TPOT and H2O AutoML, indicates that both models balanced out false negatives and positives relatively well, while Google AutoML's results indicate the model is more inclined to false negatives rather than false positives. In terms of overall performance, AutoML Tables produced the highest scoring models.

6.5.2 Functionality

Although Google AutoML Tables produced the best results with the data, functionality is still an important aspect to evaluate, especially for research purposes. TPOT's ability to export the optimal pipeline is extremely useful in research to better examine the model. The ability to easily convert TPOT pipelines to and from scikit-learn pipelines, allows models trained using TPOT easy access to all of scikit-learn's validation and evaluation tools, such as confusion matrices, easy validation results, etc. The TPOT dictionary also allows users better control over the approach TPOT takes in training the model, such as model types, feature selection and preprocessing. Also, TPOT has access to a number of open source models including most of the scikit-learn supervised learning algorithms, PyTorch neural networks and NVIDIA's GPU CUDA. An issue with TPOT is the random nature of model training making the reproduction of the optimal pipeline through TPOT difficult.

Similar to TPOT, H2O can store the optimal model produced in training, however, H2O can store several different models and the user can pick and choose which ones to evaluate, rather than just the optimal choice. H2O also provides good exploration tools within the package, unlike TPOT which relies on Pandas and scikit-learn to perform this. H2O AutoML also only makes use of a small number of techniques compared to TPOT, since all tools used are developed by the H2O.ai, where TPOT makes use of already existing algorithms. H2O does have a number of validation tools that can be used, such as its own confusion matrix algorithm. Unlike TPOT, H2O AutoML will provide the user with a feature importance list automatically. Figure 63 shows the exported table for the trained models. This provides a lot of value for the user when evaluating the produced model and the dataset's application.

AutoML Tables provides tools and functions to allow easy deployment of models, such as automatic validation results and the ability to reference a scoring threshold, which allows users to compare difference confidence levels. A comparison from this test is shown in Figure 64. Although AutoML Tables does provide the user with a confusion matrix under certain circumstances, the inability to produce one with more than 10 classes is extremely limiting. Also, the inability to see the exact contents of the model within AutoML Tables is concerning. This black box environment make model evaluation difficult and does suggest it would be difficult to make use of AutoML Tables for research. Also, the inability to adjust the split of training and testing data within the GUI is disappointing. However, AutoML Tables does provide a number of excellent tools for exploring and selecting data for training, such as the GUI based data tables. Finally, AutoML Tables provides a feature importance table for the user after training. The is shown in Figure 65.

Variable Importances:

	variable	relative_importance	scaled_importance	percentage
0	15	1.000000	1.000000	0.071874
1	26	0.807117	0.807117	0.058010
2	22	0.804953	0.804953	0.057855
3	6	0.697091	0.697091	0.050102
4	14	0.681416	0.681416	0.048976
5	23	0.670577	0.670577	0.048197
6	7	0.623159	0.623159	0.044789
7	17	0.608131	0.608131	0.043709
8	19	0.604536	0.604536	0.043450
9	27	0.585081	0.585081	0.042052
10	28	0.537057	0.537057	0.038600
11	29	0.513568	0.513568	0.036912
12	33	0.502735	0.502735	0.036133
13	9	0.484983	0.484983	0.034858
14	24	0.446478	0.446478	0.032090
15	4	0.439258	0.439258	0.031571
16	25	0.401892	0.401892	0.028885
17	32	0.398306	0.398306	0.028628
18	8	0.392690	0.392690	0.028224
19	3	0.390532	0.390532	0.028069

Figure 63: Exported H2O Feature Importance Table

6.5.3 Ease of Use

In order to evaluate ease of use, it is important to consider the end user of software. The use of GUI in Google AutoML Tables makes it the go to for users who are unfamiliar with

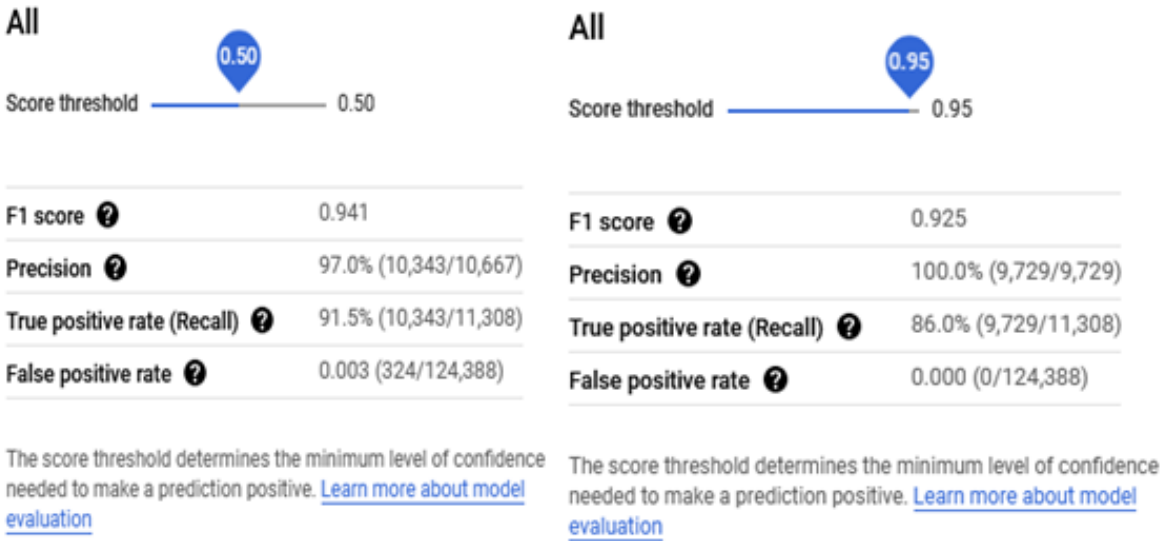


Figure 64: Comparison of Different Scoring Thresholds from AutoML Tables

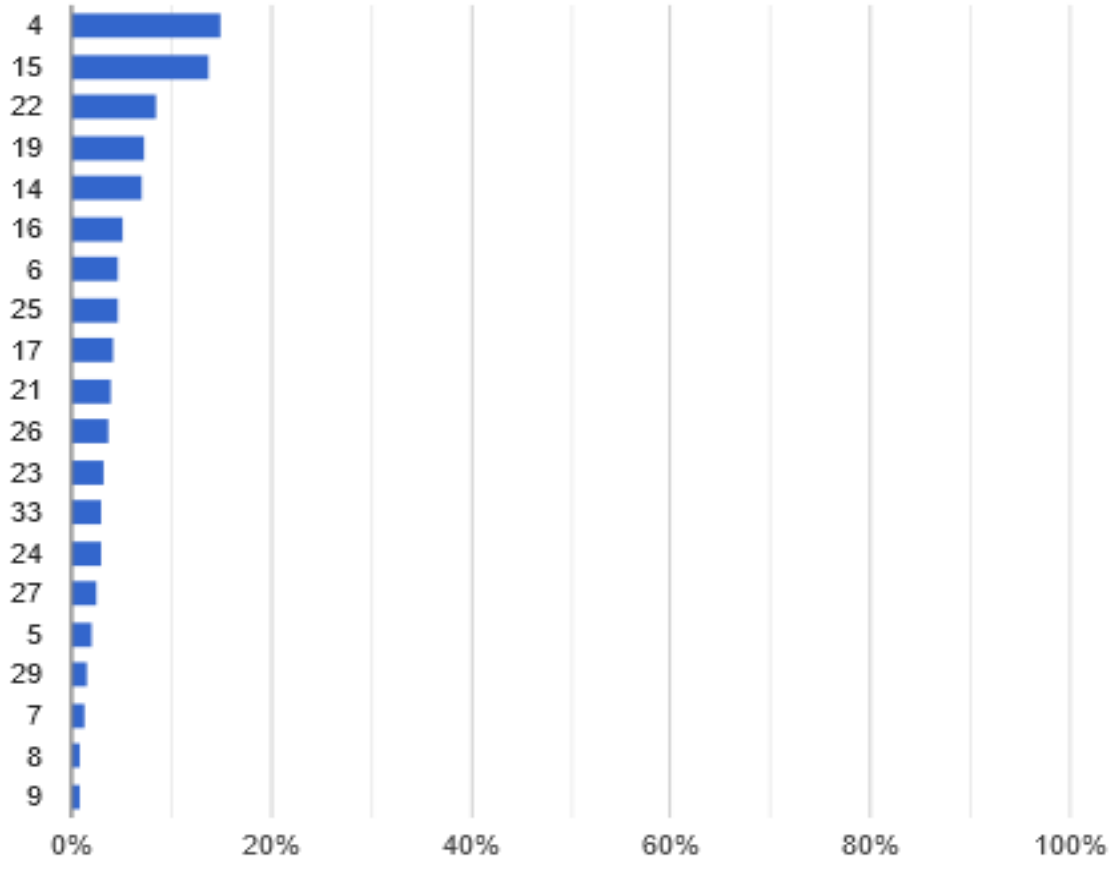


Figure 65: Feature Importance from AutoML Tables

the Python, R or programming in general. That said, researchers and experienced data scientist will have difficulty making use of the package effectively due to the black box environment, as they will be unable to examine the model at the level that will be required. Otherwise, Google AutoML Tables is very simple to use. During this experiment, the area with the most difficulty was importing the data. During the first attempt to import the data to the software, an error came up saying the column names were invalid. A header was added to the .csv file using the feature names from the GPWR, but the error persisted. In the end, it was necessary to replace the header with column numbers to get the data imported properly. The error and documentation provided little help in addressing this issue.

In terms of researcher use, both TPOT and H2O AutoML have different features that can improve ease of use. For TPOT, the functionality with scikit-learn can make the using TPOT relatively simple, as scikit-learn is a commonly used package with a high level of support in the field of data science. That said, documentation on the use of the TPOT dictionary does increase the difficulty. Also, it can be difficult to extract necessary components, such as a decision tree from pipelines if the user is unfamiliar with the concept. Also, TPOT does not currently have support for categorical features and targets which can add time to data preparation.

H2O has a number of strengths, including the ability to easily export several models and support for categorical features which can speed up data preparation. One of the biggest issues with H2O is that the AutoML instance automatically chooses if the model is classification or regression and there is no way for the user to choose. This can cause a number of issues, like ensuring the model is trained correctly. Also, H2O defaults to regression evaluation regardless of which type of model is trained. This add extra effort for evaluation and can cause confusion when the model is trained. Finally, H2O's use of its own dataframe for operations can add some additional difficulty.

6.6 Summary Remarks

All three of the AutoML methods, TPOT, H2O AutoML and AutoML Tables tested in this phase of the project, have the potential to increase the use of machine learning both, in application and research. All three were able to take the GPWR data and produce models with high validation remarks. Although there are issues with all three approaches, it is likely that efforts will be made to continue to improve each in functionality, performance and ease of use to meet the needs of user. The following lists summarize the pros and cons of each AutoML approach.

TPOT Pros

1. Large number of models available across several other machine learning packages
2. Ease of transition from TPOT to scikit-learn functions
3. Ability to easily and transparently export optimal model
4. Ability to configure customized dictionary
5. Free open source Python package

TPOT Cons

1. No support for categorical features
2. Difficult to configure customization for models
3. Stochastic in nature, making model reproduction difficult
4. Only available in Python

H2O AutoML Pros

1. Support for categorical features
2. Ability to store and export multiple models
3. All functionality availability in a single package
4. Available for use in Python, R, etc.
5. Free open source package

H2O AutoML Cons

1. Little functionality with other packages
2. Combined regression and classification instance
3. Difficult to extract model components
4. Somewhat confusion documentation

Google AutoML Tables Pros

1. Support for categorical features
2. Easy to use GUI
3. Several easy to use tools for data exploration
4. Automated evaluation
5. Easy to deploy model

Google AutoML Tables Cons

1. Paid service
2. Inability to adjust split without modifying dataset
3. Limited ability to evaluate model
4. Inability to view specifics on trained model

7 Conclusions

7.1 Future Research

7.1.1 Development of a A.I. Standard for Nuclear

The work on this project as well as other efforts around the world show the promise of data science and machine learning in the nuclear industry. However, there are a great number of barriers and challenges that must be overcome until real world application of these techniques can take place. The first major issue that needs to be addressed for actual implementation is regulatory approval. As has been mentioned, the nuclear industry in the United States is under strict regulatory control from the NRC. Any efforts to implement or rely on a machine learning based system, will have to be accepted by this body. To made progress in this area the NRC should begin developing a standard for the use of machine learning and data based systems in a nuclear reactor. This standard should establish validation benchmarks researchers can use when developing their models, such as how accurate a model must be before it can be relied on. The standard should also address the types of models that can be used, as well as the extent a model must be explained and/or visualized before it can be relied upon. Perhaps the most critical question that needs to be addressed in such a standard is the types of transients a model must be able to identify. Nuclear reactors are complex machines and as such, there are a number of things that can happen within the system. Many of these are extremely unlikely to ever occur within a system. For example, the failure rate of reactor vessel at the Dominion Surry Power Station in Virginia is one failure for every 6.7 million years[99]. It does not make sense to spend time and resources on events like these. Instead, it would be better to determine which events are most likely to happen and develop models to identify those events. The NRC already requires that all licensed commercial reactors have a detailed Probabilistic Risk Assessment (PRA) performed before a reactor can begin operations. The purpose is to identify the events at greatest risk

of occurrence. This allows for the plant to better respond in the event one of these issues occurs. The NRC should develop a probability of occurrence threshold to guide researchers. Then, researches can focus their efforts on developing models that identify the events at greatest risk of occurrence. This in turn would allow for the training of the most effective machine learning models. Selecting transient events based on a PRA of the reactor system has additional benefits. One issue with any classification system for a diagnosis is that the models can generally only classify based on known outcomes. As noted, one study proposed training a model with "I don't know option" to help inform users when the system has encountered an unknown outcome. This option has merit, but there are a number of issues that need to be considered. For example, when does the model decide it can't classify a transient? Is that criteria acceptable, and how effective is the model if it cannot diagnosis more likely transients? The use of a threshold to develop guidelines while still leaving open the possibility of encountering an unknown transient, could greatly reduce this probability. Also, it may ensure higher risk transients are more likely to be identified.

7.1.2 Human Factors When Interacting With A.I.

Another area of potential research that must occur before implementation, is the evaluation of human behavior when interacting with machine learning based systems. Automation has reduced defects and improved safety in a number of industries, but there is a tendency for users of such systems, to either become dependent on the system or find themselves unable to override the automation when the system fails. One industry that has seen this occur a number of times is the aviation industry. Due to the life or death nature of these issues in aviation, much research has been done to try and help address these human factors to automotive system interaction. A study in aviation posed a number of questions to consider when designing these kind of systems. These include: How information from the system is processed, what information is presented to the per-

son in control of the system and how decisions based on that information were implemented[100]? This approach should be applied in the nuclear industry as well, regardless if the machine learning system is used as a guide to assist operators or if the system is designed to make a preventive response. Studying how operators behave when working and interacting with these types of systems, can identity potential pitfalls and allow for better safety and accident response training. This will ensure operators are able to make decisions independent of the system if necessary. Also, if the machine learning models is going to be used as part of an autonomous system, it will ensure operators and staff have the ability to override and stop the system should it become necessary. This type of effort would likely take the form of a Human Reliability Assessment (HRA) study. Currently, there is not standard one-size fits all approach for HRA in the nuclear field, but the NRC has performed numerous studies in this area and has developed a list of good practices for this type of analysis[101]. If the implementation of a machine learning based system requires major changes to instrumentation, research has been done to produce guidelines for providing operators the best possible layout for knowledge elicitation[102].

7.1.3 Future Machine Learning Studies in Nuclear Science

As interest continues to increase in implementing machine learning for transient detection, it will be important to begin training models on different more modern designs. One of the newest designs to gain attention is the NuScale Small Modular Reactor (SMR). The first of these reactors is expected to be built at the INL in the late 2020s and the SMR concept has generated great interest. In 2021 University of Idaho, DOE and NuScale announced the opening of a new simulation lab for this new design, at the Center of Advance Energy Studies[103]. This could provide an excellent opportunity to collect data for machine learning models. Since the PRA for the NuScale design will need to be completed before the reactor begins construction, it could provide a good start for potential implementing a machine learning based diagnosis system with a modern reactor.

7.1.4 Other Approaches for Anomaly Detection

Anomaly detection will continue to be an area of high interest in the field of machine learning. As security concerns grow and the need to assure the reliability of data in real time becomes a greater concern, it is likely techniques like autoencoders will find greater use. However, there are a number of new techniques that may be applied in the future.

Perhaps the most exciting of these is the concept of generative learning, where machine learning models generate their own noise and altered data, based on what the model has learned from existing data. Training models in this manner usually takes the form of a Generative Adversarial Network (GAN). This type of machine learning approach was first proposed in 2014 by Ian Goodfellow and since, gained a lot of attention with the continued rise of neural network based models[104]. A GAN is a neural network approach where two separate neural networks are trained, similar to an autoencoder. The first network is trained to generate noisy data points based on the data. This is known as the generator. Then a separate network is trained to be able to tell the difference between the real and the noisy data. This is the discriminator network. The generator is then retrained to try and fool the discriminator and process repeats itself. Many applications of this technique have been proposed, such as image and video generation.

Although this is an exciting prospect, there are still a number of issues that need to be addressed with GANs. Geron notes that it is difficult if not impossible to determine when the models reach an equilibrium. This can cause the models to begin forgetting things they have learned and cause a sort of infinite loop. Also, validation is challenging as there is no guidance on when to start and end the evaluation. A GAN was attempted for this experiment, but evaluation of the two models could not be done at this time. A number of studies have proposed using GANs as anomaly detectors and as the concept begins to be refined and the issues addressed, it is likely the technique will show promise in the area for the future.

In addition to neural network approaches, it may be worth taking a look at SVM to

examine the GPWR data for anomalies. SVM have produced highly accurate binary classification models in many studies, including the area of anomaly detection for a number of years. One such study proposed using an unsupervised SVM model in order to better address the evaluation of outlier data[105]. This was an issue that was encountered in this study with the autoencoder approach. It may be worth comparing SVMs, autoecnoders and GANs, once the techniques have been better refined to see which techniques can better classify clean and noisy data.

7.1.5 Auto Machine Learning

One of the themes focused in this project is the use of AutoML to better streamline the machine learning training process. The expansion of neural networks and the ever increasing number of hyperparameters will ensure interest in the concept of AutoML, especially in real world applications. In this study only three AutoML approaches were evaluated. In the coming years however, it is likely others will mature and become more viable, especially as machine learning as a service begins to see more widespread use. As these approaches to AutoML release, it would be very useful to continue to evaluate the different packages to see how they compare. For example, Amazon Web Services and Meta, the parent company of Facebook, both have begun offering AutoML services to customers. As companies begin to better compete with each other, it is likely new innovations in the areas of data exploration, preparation, model training/evaluation and HPC capabilities will occur. As this happens, it would be interesting to evaluate the groups of users that may find better use of which techniques, as different users will have different needs. This will allow users to make better choices when looking at using AutoML and allow for improved efficiencies.

7.2 Final Summary

The four different experiments run for this effort help demonstrate the value that machine learning and artificial intelligence can add not only the to nuclear industry, but potentially to other fields as well. In the first effort, data was collected from the GPWR reactor simulator at CAES. This data consisted of over 30 different features from a nuclear reactor during five different events. This allowed for the creation of a dataset with over 30,000 data points. This data was then explored using the Python package Pandas to prepare the data for machine learning model training. Half of the data was used to create five machine learning models using naive Bayes classification, logistic regression, kNN classification and decision tree classification using the AutoML package TPOT. These models were extremely successful in classifying the five initial events. All five of these models scored overall validation in the high 90% range, all six models were able to perfectly distinguish normally operating reactors from those experiencing transient events, and individual transient accuracies were in the high 80 to high 90% range. In this phase of the project, the logistic regression, kNN and decision tree models performed the best, with accuracies of 98.35%, 98.55% and 98.60%. The precision, recall and f1 score also scored very high. The results from the first experiment were encouraging, as non-neural network models trained using AutoML produced high performing models. However, many questions still remained. Most concerning is whether models could be trained to identify an even larger number of transient events. To determine this, the GPWR simulator was used again to collect data on a number of simulations. This time over 168 simulations were run using more initial conditions with 12 different events. This expanded data included over 110,000 data points. Once the data was prepared using Pandas, new models were trained using TPOT. In addition to using the six approaches from the first study, it was now possible to train a model using random forest classification. The results from this study were mixed with the three naive Bayes models and logistic regression model dropping significantly in accuracy and other validation measurements. However,

the kNN and the two tree based models were able to maintain validation results in the low 90% range. That said, these three models did struggle with three transients. Those were the feedwater pump trip, turbine trip without SCRAM and electrical load rejection. The individual accuracies only reached between 60 and 70%, while the remaining 9 events scored in the high 90% range. Despite this, the results from both studies showed that non-neural network based models can produce highly accurate models in the area of reactor transients. Also, AutoML packages such as TPOT can be used to train these models. Since the models trained from the expanded GPWR dataset scored high in validation, it was possible to address a number of other concerns with the models and approach. The first of these was how would the models perform if feature data was unavailable or deemed unreliable for some reason, such as sensor failure or a security breach. To explore this impact, features were removed from the dataset based on two criteria. The features with the lowest Gini Impurity and the the features with the highest rated importance were removed. This was done for the top five features in both criteria. In total, ten additional models were trained for each of the three approaches. The results from this were positive, as TPOT was able to train models with only a small impact in validation results, around 2%. This shows that reliable models can be produced even with important features missing, increasing the redundancy of any implemented system for a reactor. This would be a high point of emphasis for any regulatory body. Another issue that needed to be addressed was the variance in the model due to changes in the random states used. TPOT relies on a stochastic process and a different random states are used in the process. To asses the impact of changing the random states on model training, the three optimal models were trained using 20 different random states. The results from this showed only a minimal impact for the three models, less than 1% change for the two tree based models and 3.32% for the kNN model. This is very encouraging as regulatory agencies would likely want to know the impact random state changes would have on results and a minimal impact suggest the results of the training and testing are relatively

stable despite the use of random numbers. Two final areas were addressed for the optimal models from the expanded dataset. First, an effort was made to improve the models. Five different data splits were used to see if more data would allow the models to better learn some of the transients that the models were having issues with. The results from this showed no major impact on validation results for the three models. The scores from this were all within the range of the variation analysis. The next attempt to improve the model was to allow TPOT more time to find a better model. To accomplish this, the population size and number of generations was increased in four different runs with the final run incrementally increasing the generations and population size to 200. This also had a minimum impact on model performance, as all results were still within the validation range. Since there was no noticeable improvement in the models, it was necessary to try and identify why the models struggled with the three transients. Back tracing along the a given decision tree would not necessarily provide a great deal of insight into the issues. This was due to the number of preprocessing steps a dataset undergoes before training. Instead, a decision tree model was taken from the variation analysis and the data that was misclassified was separated into six sets. The descriptive statistics were calculated for each set and compared to the correct classifications for the three transients. Analysis of the statistics showed a great deal of variation in both levels for steam generator. Many of the misclassified points showed more similarity in this feature to the transient they were incorrectly classified as. This provided strong evidence that this is the issue behind misclassification.

The third phase of this project involved using machine learning to detect anomalies within the data collected from the GPWR simulator. Having a machine learning model to filter the data coming from the reactor and detect data that is significantly different from what was expected can have a number of applications, especially in nuclear safety. In some situations, it can be a final line of defense in a cyber attack, but it can also be used to help operators identify sensors that are malfunctioning, allowing for quicker repairs and

more reliable instrumentation systems. In this experiment a portion of the data collected from the expanded GPWR dataseet was split and randomly altered at different levels between 0.5 and 2 standard deviations from the mean. An autoencoder neural network was built to try identify these altered points. Using a threshold calculated from the data, it was found that the model could detect all altered points at the high levels of noise. The model did have issues identifying about a quarter of the true data as an anomaly. Also, as the noise level decreased, the model's performance decreased. At the lowest level, the model struggled distinguishing between altered and unaltered data, but this was expected. As innovation in anomaly detection continues, the development and implementation of these techniques for nuclear applications, such as reactor instrumentation monitoring, could contribute greatly to system security and reliability. The final phase of this project was to compare and contrast TPOT and two other AutoML approaches. H2O AutoML and Google AutoML Tables. The GPWR data was used to train classification models using the most basic options for each approach. Each model was allowed to run for three hours in a cloud computing environment. Once the model were trained, all three approaches were evaluated based on performance, functionality and ease of use. In terms of performance, H2O and TPOT compared very well with each producing a model using XGBoost with accuracy measurements around 93%. While AutoML Tables did not provide an accuracy measurement, the precision of 97% suggests a highly accurate model. All three models scored within the range in recall.

In terms of functionality, all three packages had their strengths and weaknesses. AutoML Tables provides excellent exploration tools and evaluation visualizations, but due to the black box environment, it is difficult to get specifics. TPOT has access to a large number of models from different packages, but it does rely completely on other software and can only export one model. Finally, H2O has a large number of data exploration/evaluation tools and the leader board allows for users to examine a large number of models, but the selection of models is much more limited than TPOT.

Finally, in terms of ease of use, AutoML Tables is by far the easiest to use due to its full use of a GUI and export capabilities, which allows access to those with limited programming experience. TPOT should be easy to use for anyone familiar with scikit-learn and this makes it a good entry point for those with a Python background. That said, model customization can be challenging due to lack of documentation and there are still challenges with the CUDA GPU and PyTorch extensions. H2O is similar to scikit-learn and many functionalities that are easy to learn for those familiar with Python. That said, the AutoML instances can be difficult to use since it chooses between regression and classification and the user has no control over the selection. This can result in very confusing validation results, especially in the area of classification.

The results from this project have helped to demonstrate the potential of machine learning and AutoML in the nuclear industry. Nuclear requires some of the most rigorous safety controls of any industry and the implementation machine learning could help in the perception of the field. These concepts will continue to improve and develop in the coming years and as a result and will be more widely used in daily life. It is hoped that this research will help contribute to both, nuclear safety and data science.

References

[1] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow 2nd Edition*. O'Reilly. September 2019.

[2] *The economic impact of artificial intelligence on the UK economy*. PWC. 2017.

[3] *Reactor analytics drives nuclear industry towards machine learning*. Nuclear Energy Insider. November 15th, 2017.

[4] Kislay Keshari. *Top 10 Applications of Machine Learning : Machine Learning Applications in Daily Life*. Edureka, https://www.edureka.co/blog/machine-learning-applications/. November 2020.

[5] Bernard Marr. *The Amazing Ways Coca Cola Uses Artificial Intelligence And Big Data To Drive Success*. Forbes, https://www.forbes.com/sites/bernardmarr/2017/09/18/ the-amazing-ways-coca-cola -uses-artificial-intelligence-ai-and-big-data-to-drive-success/. September 2017.

[6] Michael Martinez. *Amazon: Everything you wanted to know about its algorithm and innovation*. IEEE Computing Society, https://www.computer.org/ publications /tech-news/trends/amazon-all-the-research-you-need-about-its-algorithm-and-innovation. 2017.

[7] J. O. Awoyemi, A. O. Adetunmbi and S. A. Oluwadare. *Credit card fraud detection using machine learning techniques: A comparative analysis*. International Conference on Computing Networking and Informatics (ICCNI), Lagos, Nigeria, 2017, pp. 1-9, doi: 10.1109/ICCNI.2017.8123782. 2017.

[8] *Who is using scikit-learn?* Scikit-learn developers, https://scikit-learn.org/stable/testimonials/testimonials.html# who-is-using-scikit-learn.

[9] Bill Siwicki. *Machine learning helps cancer center with targeted COVID-19 outreach.* HealthcareITNews,https://www.healthcareitnews.com/news/machine-learning-helps-cancer-center-targeted-covid-19-outreach. February 2021.

[10] Zeki Murat Çınar, Abubakar Abdussalam Nuhu, Qasim Zeeshan, Orhan Korhan, Mohammed Asmael and Babak Safaei. *Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0.* Sustainability. 2020.

[11] *Using AI-Powered Visual Inspection to Spot Defects." IBM MediaCenter.* IBM,mediacenter .ibm.com /media/ Using+AI-powered+visual+ inspection +to+spot+defects%20 /0_11oe6sng.

[12] Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, Arun Hampapur. *Improving rail network velocity: A machine learning approach to predictive maintenance.* Transportation Research Part C: Emerging Technologies, Volume 45, Pg 17. 2014.

[13] Zhiyi Wang, Jiachen Zhong, Jingfan Li and Cui Xi. *Fault diagnosis of air-conditioning refrigeration system based on sparse autoencoder.* International Journal of Low Carbon Technologies, Vol. 14 Issue 4, p487. December, 2019.

[14] Knief Ronald. *Nuclear Engineering: Theory and Technology of Commercial Nuclear Power 2nd Edition.* American Nuclear Society. 2014.

[15] Hopkins Andres. *Was Three Mile Island a "Normal Accident"?* Journal of Contingencies and Crisis Management, Vol. 9 Issue 2 Pg. 67. 2001.

[16] Chen WL. *Simulation for training and decision-making in large-scale control systems: Part 6: Power plant simulators.* SIMULATION., 1980;35(4):133-136. 1980.

[17] *NUCLEAR POWER PLANT SIMULATION FACILITIES FOR USE IN OPERATOR TRAINING, LICENSE EXAMINATIONS, AND APPLICANT EXPERIENCE REQUIREMENTS.* U.S. NUCLEAR REGULATORY COMMISSION. April, 2011.

[18] Kenneth Thomas. Ronald Boring Julius Persensky. *Deployment of a Full-Scope Commercial Nuclear Power Plant Control Room Simulator at the Idaho National Laboratory.* Idaho National Laboratory. 2011.

[19] *VERIFICATION, VALIDATION, REVIEWS, AND AUDITS FOR DIGITAL COMPUTER SOFTWARE USED IN SAFETY SYSTEMS OF NUCLEAR POWER PLANTS.* U.S. NUCLEAR REGULATORY COMMISSION. July 2013.

[20] *Use of control room simulators for training of nuclear power plant personnel.* International Atomic Energy Agency. September 2003.

[21] F. Chen and M. R. Jahanshahi. *NB-CNN: Deep Learning-Based Crack Detection Using Convolutional Neural Network and Naïve Bayes Data Fusion.* IEEE Transactions on Industrial Electronics, Vol. 65, Pg 4392. May 2018.

[22] S. Million-Weaver. *Eagle-eyed machine learning algorithm outdoes human experts.* University of Wisconsin Madison. July, 2018.

[23] J. Bae, A. Rykhlevskii, G. Chee, K. Huff. *Deep learning approach to nuclear fuel transmutation in a fuel cycle simulator.* Annals of Nuclear Energy, Vol. 139.

[24] A.Erdo gana, M. Geckinlib. *A PWR reload optimisation code (XCore) using artificial neural18networks and genetic algorithms.* Annals of Nuclear Energy Vol. 39.

[25] H. Vella. *What Will the Power Plant of the Future Look Like?* General Electric. April, 2018.

[26] Y. Zeng, J. Liu, K. Sun, L. Hu. *Machine learning based system performance prediction model for reactor control.* Annals of Nuclear Energy, Volume 113, Pg. 270.

[27] X. Liab, X. Fu, F. Xiong, X.Bai. *Deep learning-based unsupervised representation clustering methodology for automatic nuclear reactor operating transient identification.* Knowledge-Based Systems, Vol.204.

[28] YO-IMORU, R. M.; CILLIERS, A. C. *Continuous machine learning for abnormality identification to aid condition-based maintenance in nuclear power plant.* Annals of Nuclear Energy, Vol. 118, Pg. 61.

[29] R. Gronberg. *AI could one day control nuclear reactors; NC State researchers could make it happen.* News Observer. June 2018.

[30] Lin L. , Bao H., Dinh N. *On the Formalization of Development and Assessment Process for Digital Twins.* Transactions of the American Nuclear Society Vol. 123, Pg 268. 2020.

[31] M. Kim, etc. *NN-Based online anomaly detection in nuclear reactors for highly imbalanced datasets with uncertainty.* Nuclear Engineering & Design. Vol. 364.

[32] Y. Yuantao, etc. *mall-batch-size convolutional neural network based fault diagnosis system for nuclear energy production safety with big-data environment.* International Journal of Energy Research, Vol. 44, 5841.

[33] Y. Chen, M. Narita, T.Yamada. *Nuclear reactor diagnostic system using genetic algorithm (GA)-trained neural networks.* Electrical Engineering in Japan, Vol 115, 88. 1995.

[34] S Cheon, S. Chang. *Application of Neural Networks to a Connectionist Expert System for Transient Identification in Nuclear Power Plants.* Nuclear Technology, Vol 102, Pg. 177.

[35] M. dos Santos, etc. *Deep rectifier neural network applied to the accident identification problem in a PWR nuclear power plant.* Annals of Nuclear Energy, Volume 133, 400.

[36] T. Kim, J.Park, B.Lee, S. Seong. *Deep-learning-based alarm system for accident diagnosis and reactor state classification with probability value.* Annals of Nuclear Energy, Volume 133, Pg. 723.

[37] D. Chang, M. Liu, Y. Lee. *Accident diagnosis of a PWR fuel pin during unprotected loss of flow accident with support vector machine*. Nuclear Engineering and Design, Volume 352, 110184.

[38] C. Gottlieb, V. Arzhanov, W. Gudowski, N. Garis. *FEASIBILITY STUDY ON TRANSIENT IDENTIFICATION IN NUCLEAR POWER PLANTS USING SUPPORT VECTOR MACHINES*. Nuclear Technology, Volume 155, Pg. 67.

[39] *Electric Power Monthly with Data for May 2020*. Energy Information Administration. July 2020.

[40] Amjith Ramanujam, Ellen Livengood. *Python at Netflix*. https://netflixtechblog.com/python-at-netflix-bba45dae649e. April,2019.

[41] Thomas. Oliphant. *Guide to NumPy*. December 2006.

[42] McKinney Wes. *"pandas: powerful Python data analysis toolkit Release 0.24.2."*. March 13 2019.

[43] Mckinney Wes. *"pandas: a Foundational Python Library for Data Analysis and Statistics"*. 2011.

[44] Scikit-learn Developers. *"About Us"*. Scikit-learn.org.https://scikit-learn.org/stable/about.html#people.

[45] Scikit-learn Developers. *"Who is using scikit-learn?"*. Scikit-learn.org https://scikit-learn.org/stable/testimonials/testimonials.html.

[46] etc all Pedregosa. *scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830. 2011.

[47] *"Scikit-Learn User Guide Release 0.21.2"*. Scikit-learn.org. May 2019.

[48] Alphabet. *"Cloud AutoML"*. https://cloud.google.com/automl/docs/.

[49] Epistassislabs. *"Home-TPOT"*. https://epistasislab.github.io/tpot/.

[50] R. Olson, N. Bartley, R. Urbanowicz, & Moore. *Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science*. GECCO, pg 485. 2016.

[51] J. Darbon Smith L. Likforman-Sulem. *Effect of Pre-Processing on Binarization*. Boise State University. January 1, 2010.

[52] Scikit-learn Developers. *5.3. Preprocessing data*. https://scikit-learn.org/stable /modules/preprocessing.html#.

[53] Scikit-learn Developers. *Compare the effect of different scalers on data with outliers*. https://scikit-learn.org/stable/auto_examples /preprocessing/ plot_all_scaling.html #sphx-glr-auto-examples- preprocessing-plot-all-scaling-py.

[54] B. Recht A. Rahimi. *Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning*. UC Berkeley.

[55] Scikit-learn Developers. *sklearn.cluster.FeatureAgglomeration*. https://scikit-learn.org /stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html.

[56] Scikit-learn Developers. *"2.3. Clustering"*. https://scikit-learn.org/stable/modules /clustering.html#hierarchical-clustering.

[57] L. Smith. *A Tutorial on Principal Component Analysis*. University of Montreal. February 26th 2002.

[58] Scikit-learn Developers. *2.5. Decomposing signals in components*.

[59] P. Breheny. *Family Wise Error Rates*. University of Iowa. January 25th, 2016.

[60] Scikit-learn Developers. *Naive Bayes*. https://scikit-learn.org/stable/modules /naive _bayes.html.

[61] H. Zhang. *The Optimality of Naïve Bayes*. University of New Brunswick.

[62] J. Eberhardt. *Bayesian Spam Detection*. University of Minnesota Morris.

[63] P. Rai. *Supervised Learning: K-Nearest Neighbors and Decision Trees*. University of Utah. August 25, 2011.

[64] Scikit-learn Developers. *1.6. Nearest Neighbors*. https://scikitlearn.org/stable/ modules/neighbors.html#.

[65] T. Mitchel. *GENERATIVE AND DISCRIMINATIVE CLASSIFIERS: NAIVE BAYES AND LOGISTIC REGRESSION*. Carnegie Mellon/McGraw Hill. September 23 2017.

[66] Scikit-learn Developers. *1.1. Generalized Linear Models*. https://scikit-learn.org/stable /modules/ linear_model.html#logistic-regression.

[67] R. Mitchel. *Decision Tree Learning*. Princeton University.

[68] Scikit-learn Developers. *1.10. Decision Trees*. https://scikit-learn.org/stable/ modules/tree.html.

[69] Scikit-learn Developers. *3.3. Model evaluation: quantifying the quality of predictions*. https://scikit-learn.org/stable/modules/model_evaluation.html#.

[70] *GENERIC PWR SIMULATOR Training Guide*. Western Service Cooperation. May 2017.

[71] N. Siu. *Software Verification and Validation: Examples from the Safety Arena*. U.S. Nuclear Regulatory Commission Office of Nuclear Regulatory Research. September 2015.

[72] NRC. *NUREG-0800, 15.0 INTRODUCTION - TRANSIENT AND ACCIDENT ANALYSES*. 2007.

[73] WCS. *GENERIC PWR SIMULATOR Major Transients Report*. June 2014.

[74] Robert M. Bell, Yehuda Koren and Chris Volinsky. *The BellKor solution to the Netflix Prize*. AT&T Labs – Research.

[75] Scikit-learn Developers. *1.11. Ensemble methods*. https://scikit-learn.org/stable/modules /ensemble.html#forest.

[76] Scikit-learn Developers. *sklearn.ensemble.RandomForestClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[77] Alexis Perrier. *Feature Importance in Random Forests*. https://alexisperrier.com/datascience/2015/08/27/feature-importance-random-forests-gini-accuracy.html. 2015.

[78] Scikit-learn. *3.1. Cross-validation: evaluating estimator performance*. https://scikit-learn.org/stable/modules/cross_validation.html.

[79] *ELECTRICITY GRID CYBERSECURITY DOE Needs to Ensure Its Plans Fully Address Risks to Distribution Systems*. Government Accountability Office, GAO-21-81. March 2021.

[80] NRC. *REGULATORY GUIDE 5.71 (New Regulatory Guide) CYBER SECURITY PROGRAMS FOR NUCLEAR FACILITIES*. 2010.

[81] Pierre Baldi. *Autoencoders, Unsupervised Learning, and Deep Architectures*. Proceedings of ICML Workshop on Unsupervised and Transfer Learning, vol 27., Pg 37-40. 2012.

[82] all Martin Abadi etc. *TensorFlow: A system for large-scale machine learning*. 12th USENIX symposium on operating systems design and implementation, pp. 265-283. 2016. 2016.

[83] TensorFlow. *TensorFlow 2.0 is now available!* Medium.com, https://medium.com/tensorflow/ tensorflow -2-0-is-now-available-57d706c2a9ab. 2019.

[84] TensorFlow. *Why TensorFlow?* https://www.tensorflow.org/about.

[85] Keras. *About Keras*. https://keras.io/about/.

[86] K. Han, etc, all. *AUTOENCODER INSPIRED UNSUPERVISED FEATURE SELECTION*. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) pp. 2941-2945. 2018.

[87] Antonio L. Alfeo, etc, all. *Using an autoencoder in the design of an anomaly detector for smart manufacturing*. Pattern Recognition Letters, Volume 136, Pg 272-278. 2020.

[88] A. Gokhan Yavuz R. Can Aygun. *Network Anomaly Detection with Stochastically Improved Autoencoder Based Models*. 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing. 2017.

[89] Thiée, L.-W. *A systematic literature review of machine learning canvases*. Gesellschaft für Informatik, Bonn. (S. 1221-1235). DOI: 10.18420/informatik2021-101. 2021.

[90] H2O.ai. *H2O.ai Leadership*. https://www.h2o.ai/company/team/leadership-team/.

[91] LeDell E, Poirier S. *H2o automl: Scalable automatic machine learning*. In Proceedings of the AutoML Workshop at ICML 2020. 2020.

[92] Alphabet. *AutoML Products*. https://cloud.google.com/automl/docs.

[93] Alphabet. *AutoML Tables documentation*. https://cloud.google.com/automl-tables/docs.

[94] Alena Orlenko et al. "Model selection for metabolomics: predicting diagnosis of coronary artery disease using automated machine learning". In: *Bioinformatics* 36.6 (Nov. 2019), pp. 1772–1778. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz796. URL: https://doi.org/10.1093/bioinformatics/btz796.

[95] Jessica Dafflon et al. "An automated machine learning approach to predict brain age from cortical anatomical measures". In: *Human Brain Mapping* 41.13 (2020), pp. 3555–3566. DOI: https://doi.org/10.1002/hbm.25028. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.25028.

[96] Gomathi S. et al. "Pattern Analysis: Predicting Covid-19 Pandemic in India Using AutoML". In: *World Journal of Engineering* ahead-of-print (Nov. 2020). DOI: 10.1108/WJE-09-2020-0450.

[97] Alexandre L.A Lopes V. “Auto-Classifier: A Robust Defect Detector Based on an AutoML Head”. In: *Lecture Notes in Computer Science* (2020). DOI: https://doi.org/10.1007/978-3-030-63830-6_1.

[98] Nurul Afiqah and Nur Anida Jumadi. “Leg Flexibility Classification Using AutoML Tables”. In: 1 (Dec. 2020), pp. 270–279. DOI: 10.30880/eeee.2020.01.01.032.

[99] S. Levine. *RESEARCH INFORMATION LETTER - ID, PRESSURE VESSEL FAILURE PROBABILITY PREDICTION*. Office of Nuclear Regulatory Reserach. 1977.

[100] Valerie Gawron. *Automation in Aviation—Accident Analyses*. Center for Advanced Aviation System Development, MITRE. January 2019.

[101] Sandia National Lab Nuclear Regulator Commission. *Good Practices for Implementing Human Reliability Analysis (HRA)*. NUREG-1792. 2005.

[102] R. Boring, etc, all. “Guideline for Operational Nuclear Usability and Knowledge Elicitation ”. In: (2015). Procedia Manufacturing, Volume 3.

[103] Office of Nuclear Energy. *NuScale SMR Simulation Lab Opens in Idaho*. https://www.energy.gov /ne/articles/ nuscale-smr-simulation-lab-opens-idaho. 2021.

[104] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[105] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. “Enhancing one-class support vector machines for unsupervised anomaly detection”. In: *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. 2013, pp. 8–15.

Appendix A Publications from Research

1. Reactor Transient Classification Using Machine Learning, Transaction of the American Nuclear Society Vol. 121, Winter 2019.
2. Nuclear Reactor Transient Diagnostics using Classification and AutoML, Nuclear Technology, Volume 208 Issue 2
3. Expanded Analysis of AutoML Models for Nuclear Transient Classification, Nuclear Engineering and Design. (Paper accepted February, 2022, publication pending.)

Appendix B Python Packages Used

Table 44: Model Training Expanded Dataset

Python Package	Initial Version Used
Python	3.7.6
NumPy	1.18.1
Pandas	1.0.1
Scikit Learn	0.22.1
TPOT	0.9.5

Table 45: Auto Encoder Study

Python Package	Initial Version Used
Python	3.7.12
NumPy	1.19.5
Pandas	1.1.5
Scikit Learn	1.0.1
TensorFlow	2.7.0

Table 46: AutoML Study

Python Package	Initial Version Used
Python	3.7.6/3.7.12
NumPy	1.19.5
Pandas	1.1.3/1.1.5
Scikit Learn	1.0.2
TPOT	0.11.5
H2O	3.36.0.2

Appendix C Optimal Tree Output

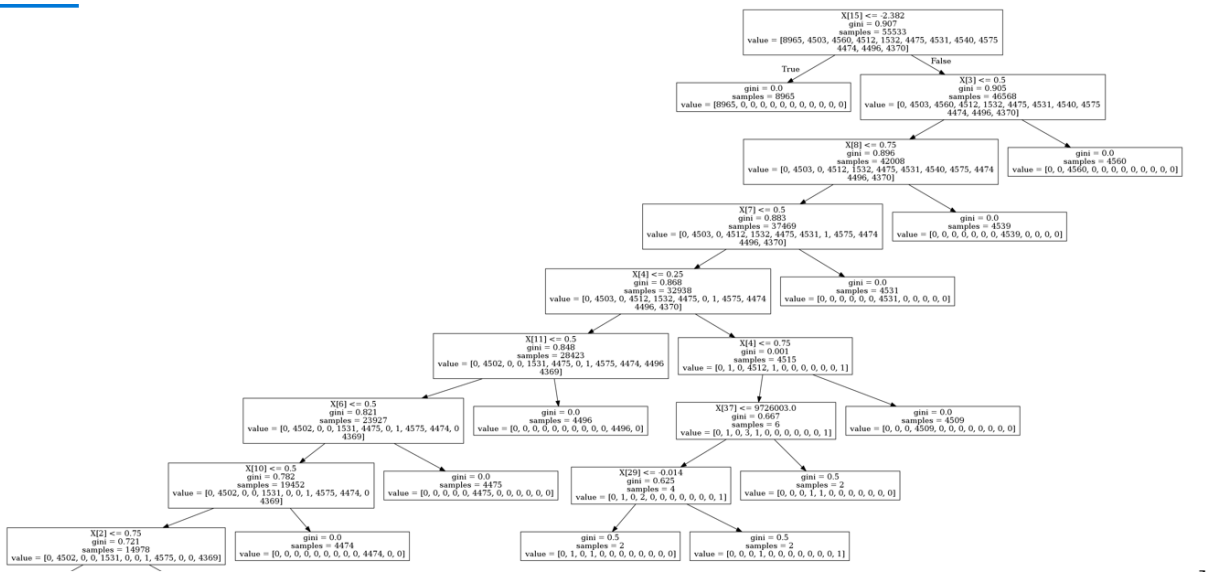


Figure 66: Optimal DT Section 1

/

Figure 67: Optimal DT Section 2

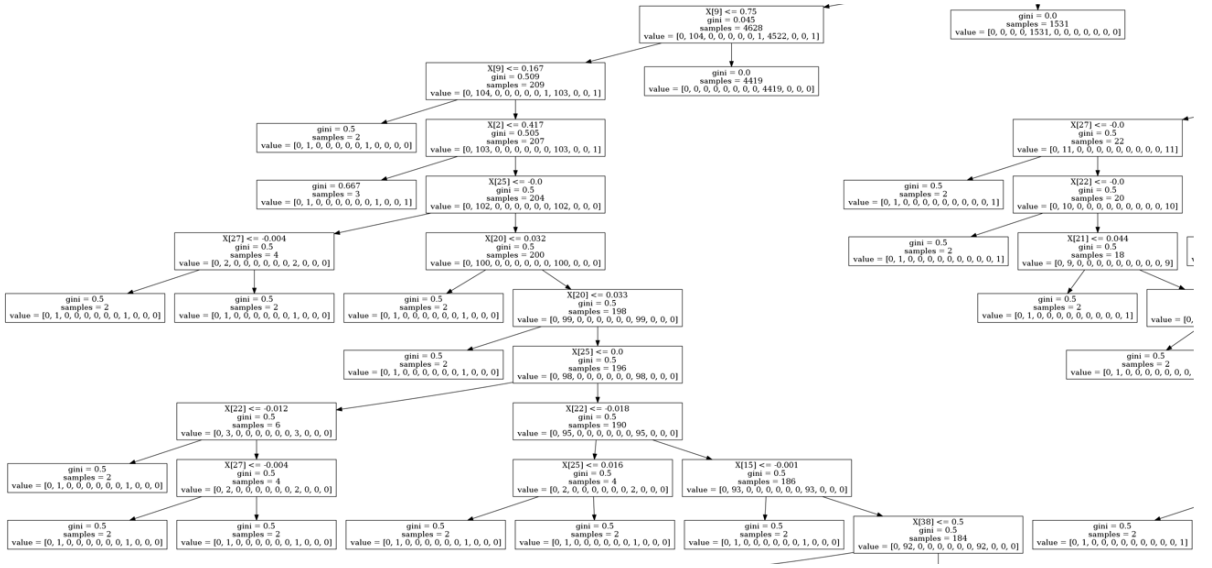


Figure 68: Optimal DT Section 3

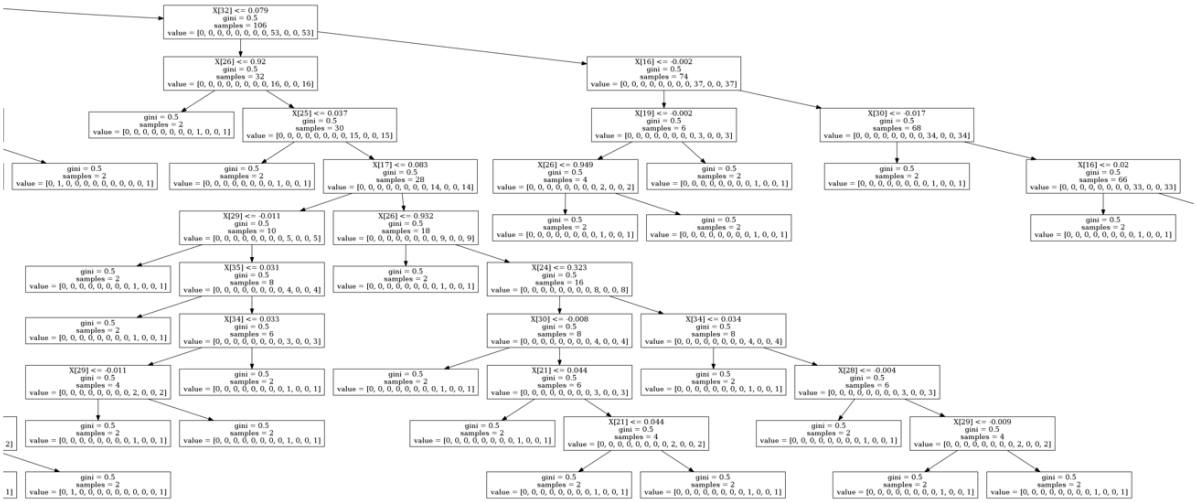


Figure 69: Optimal DT Section 4

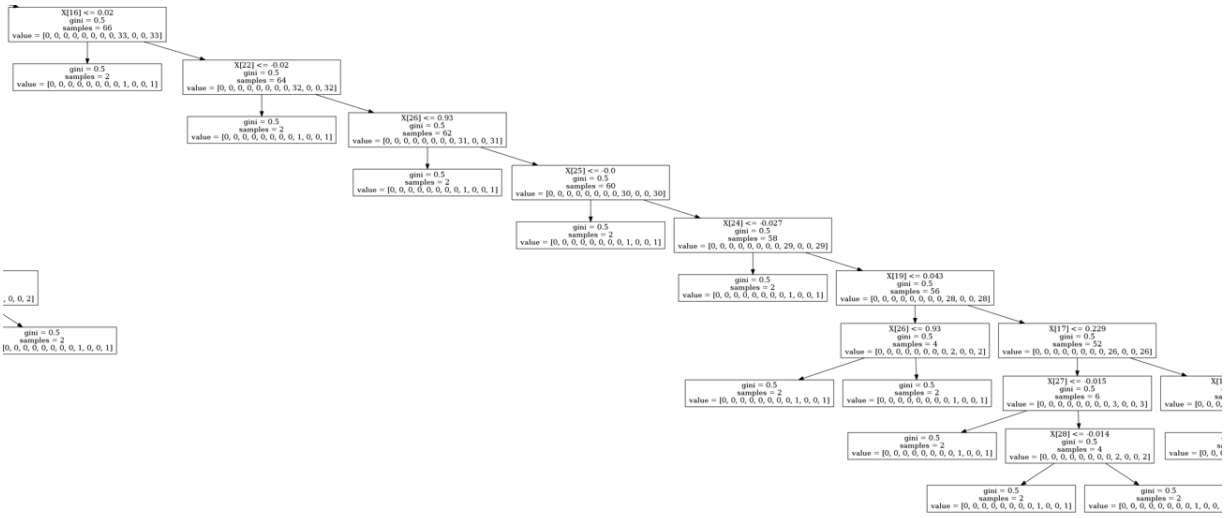


Figure 70: Optimal DT Section 5

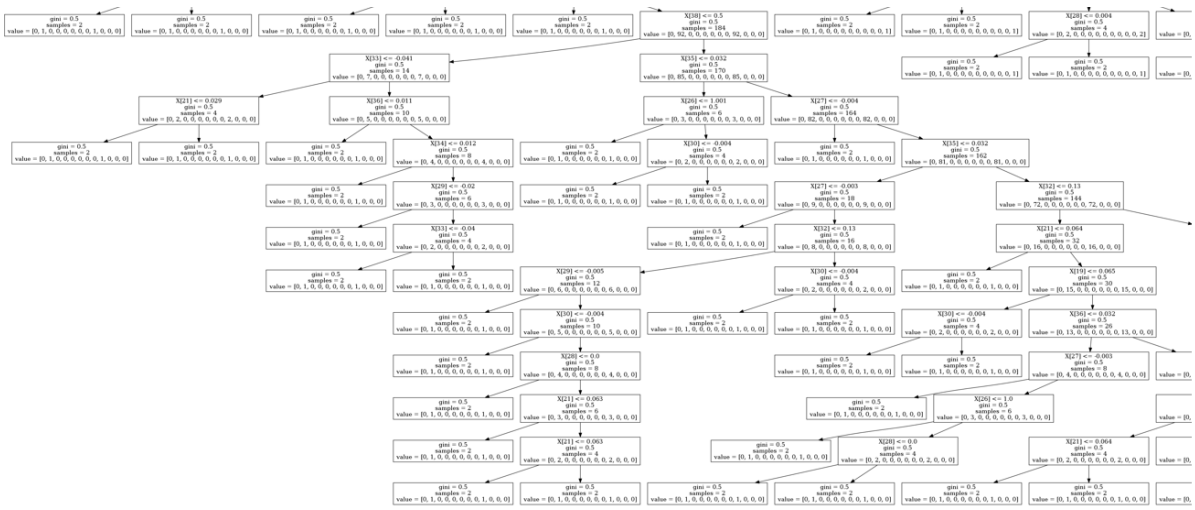


Figure 71: Optimal DT Section 6

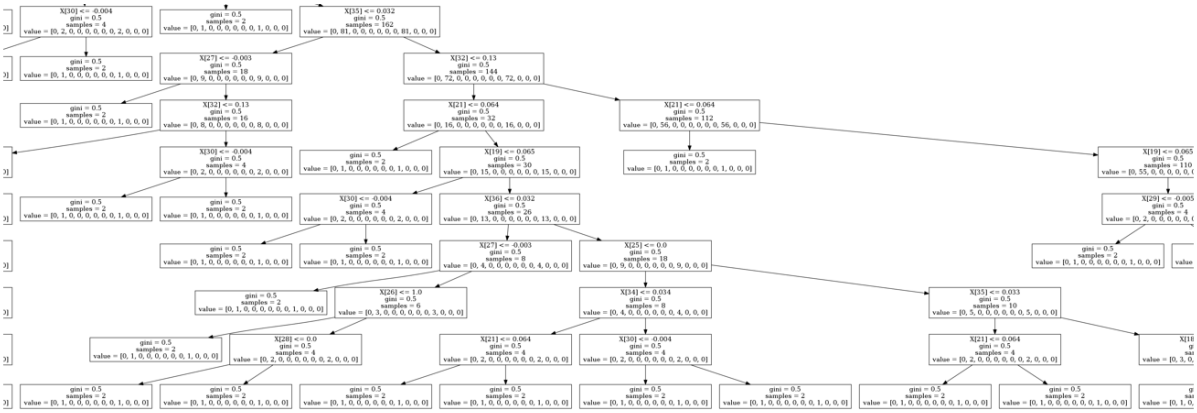


Figure 72: Optimal DT Section 7

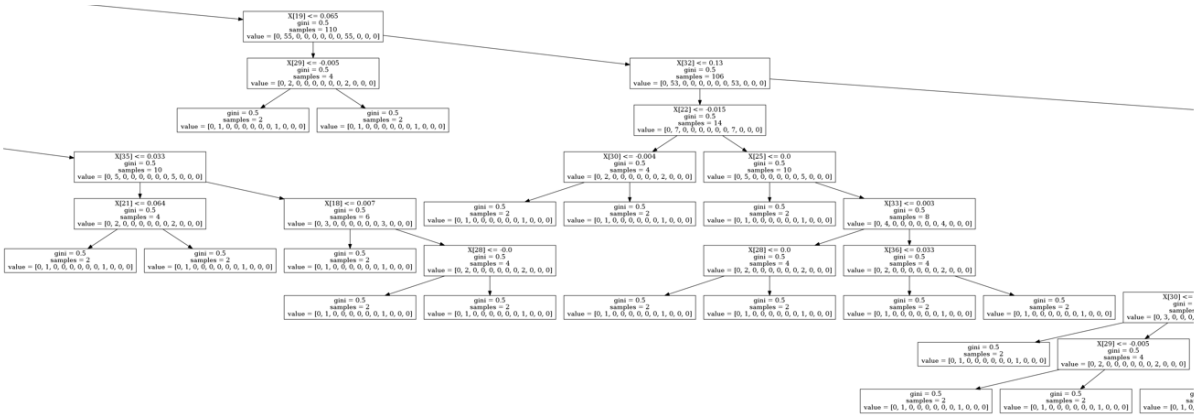


Figure 73: Optimal DT Section 8

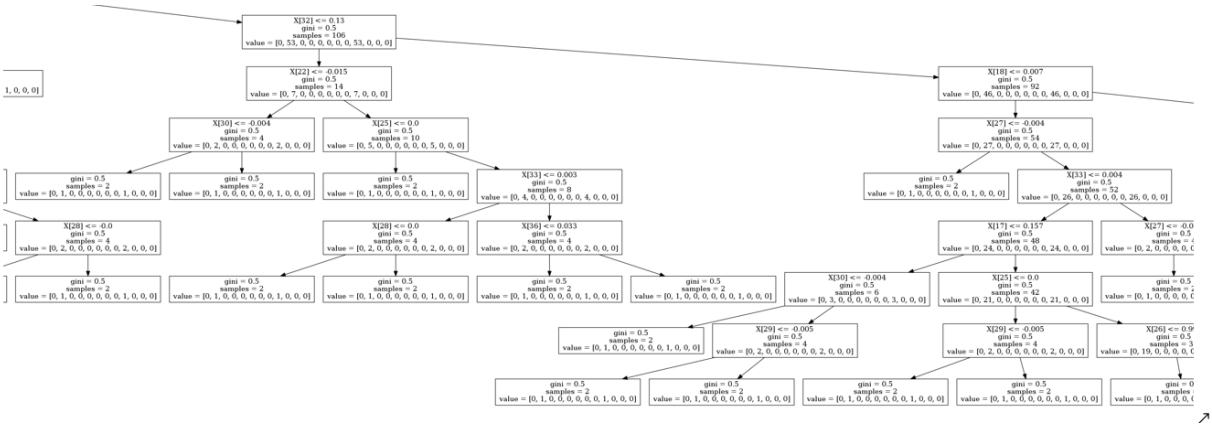


Figure 74: Optimal DT Section 9

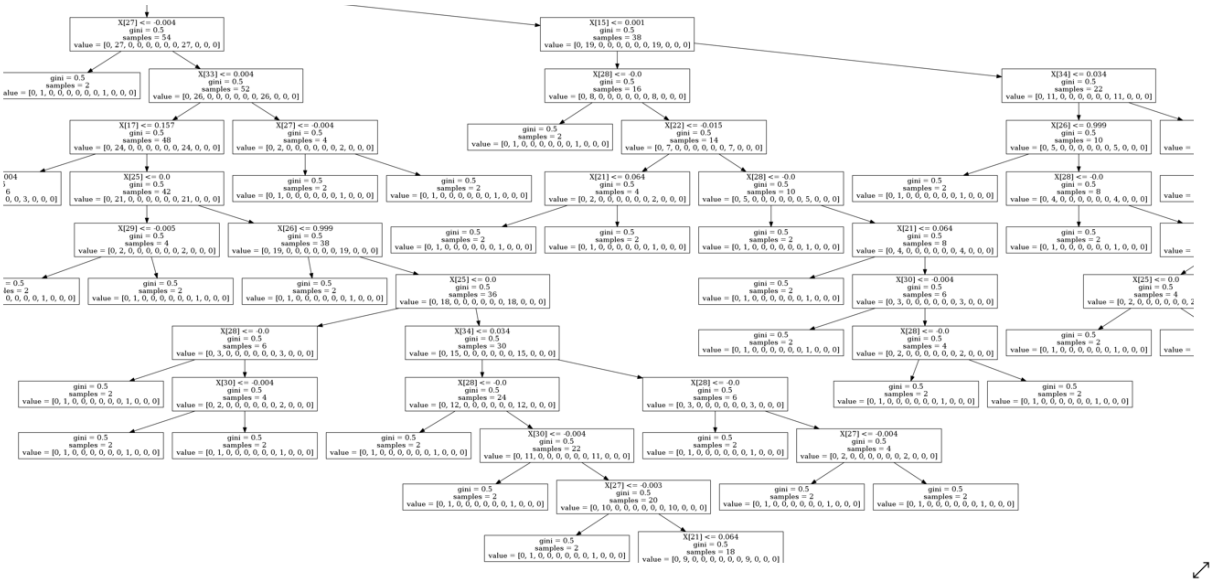


Figure 75: Optimal DT Section 10

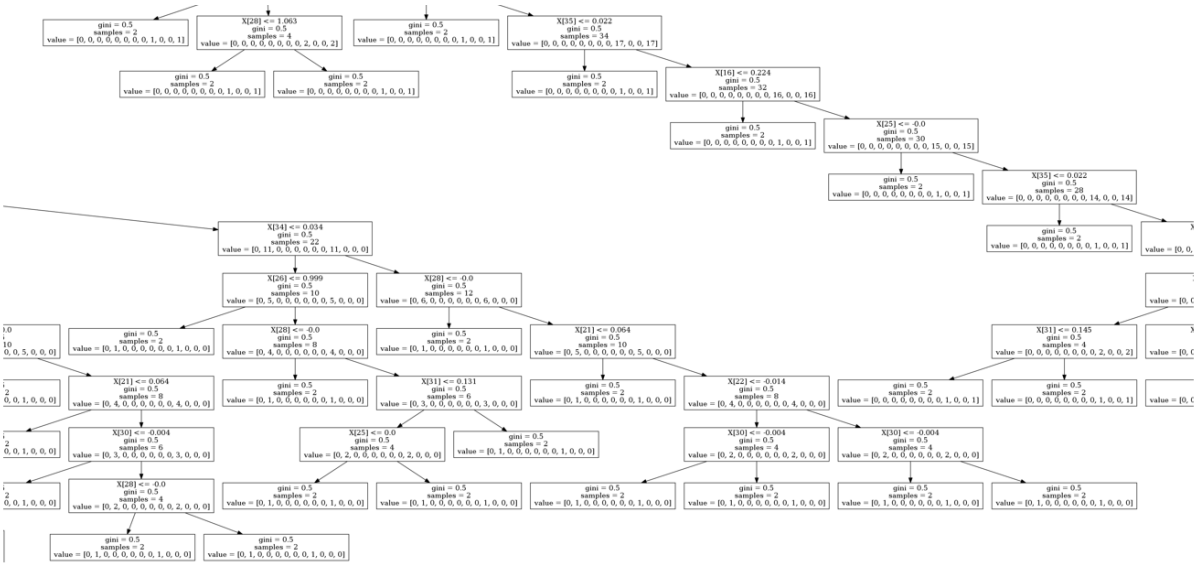


Figure 76: Optimal DT Section 11

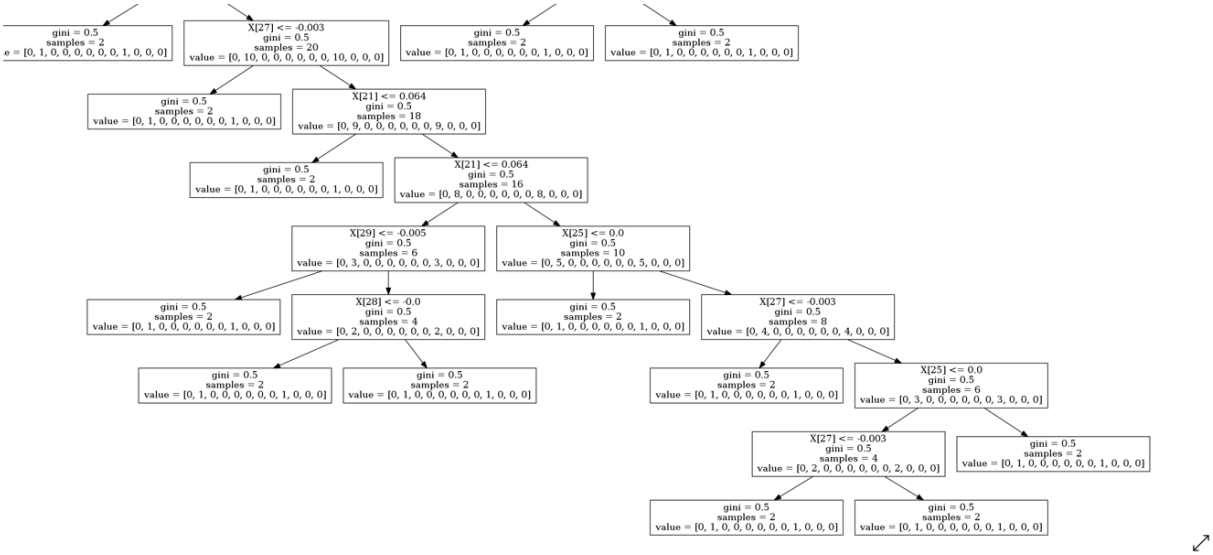


Figure 77: Optimal DT Section 12

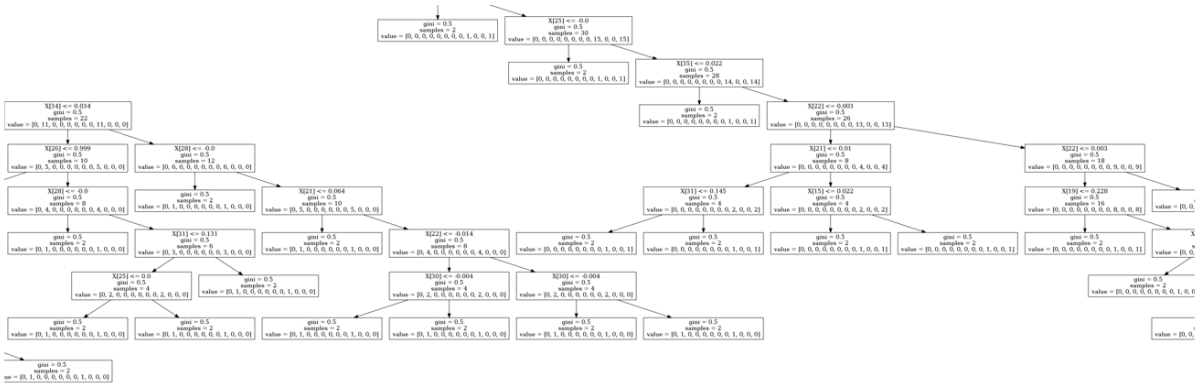


Figure 78: Optimal DT Section 13

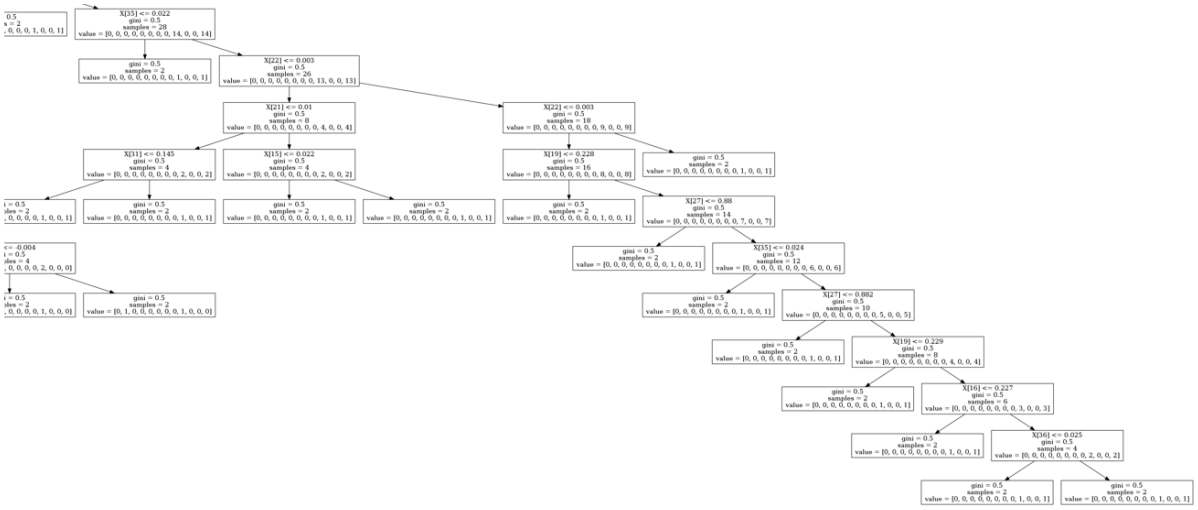


Figure 79: Optimal DT Section 14

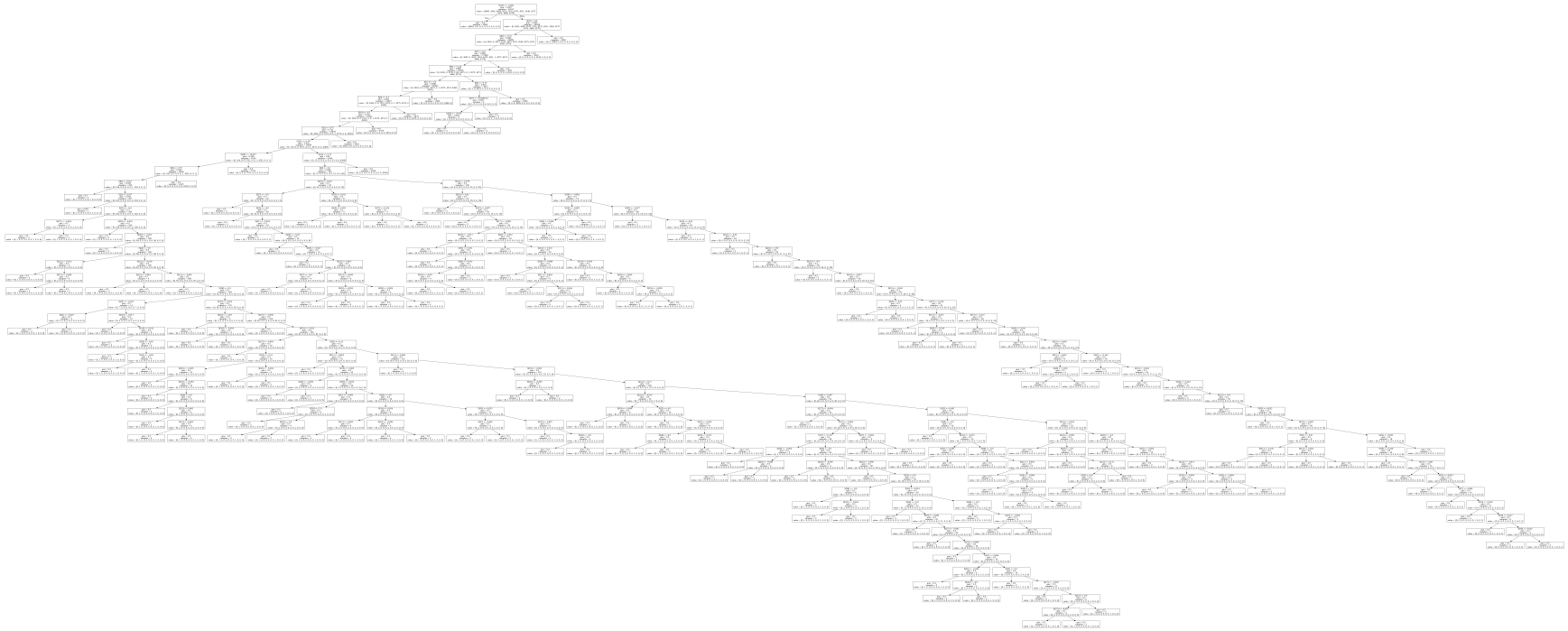


Figure 80: Full Optimal DT

Appendix D Process Visualization

```
# Average CV score on the training set was: 0.9250355610105677
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectFwe
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.tree import DecisionTreeClassifier
from tpot.export_utils import set_param_recursive
exported_pipeline = make_pipeline(
    SelectFwe(),
    RobustScaler(),
    DecisionTreeClassifier()
)
```

Figure 81: Re-create Optimal Tree From TPOT

```
     RCS LVL LOOP 1 WR  RCS LVL LOOP 1 NR  HOT LEG 1 TEMPERATURE  \
503               2.4                4.5                564.915

     COLD LEG 1A “TEMPERATURE  COLD LEG 1B TEMPERATURE  HOT LEG 2 TEMPERATURE  \
503                  564.197                  564.197                564.914

     COLD LEG 2A TEMPERATURE  COLD LEG 2B TEMPERATURE  PZR SURGE LINE TEMP  \
503                  564.197                  564.197              636.156

     PORV DISCH PZR TEMPERATURE  CONTAINMENT PRESSURE  \
503                     108.003                  0.0

     CONTAINMENT TEMPERATURE  SG-1 NR LEVEL  SG-2 NR LEVEL  FW FLOW TO SG-2  \
503                  110.166        54.0041        26.5699           11.739

     MS FLOW FROM SG-1 LINE-1A  MS FLOW FROM SG-1 LINE-1B  \
503                     11.739                  11.7256

     MS FLOW FROM SG-2 LINE-2A  MS FLOW FROM SG-2 LINE-2B  SG-2 PRESSURE  \
503                    11.7256                  1142.65        1142.65

     CALCULATED AVERAGE TEMPERATURE  PRESSURIZER PRESSURE  \
503                        564.556               2236.64

     PRESSURIZER WATER TEMPERATURE  PRESSURIZER STEAM TEMPERATURE  \
503                        652.808                        652.808

     GENERATOR POWER EOL MOL
503             0.0   1   0
(1, 27)
```

Figure 82: Raw Data to Test

```
[126]: #Transform data using pipeline
X_transformed= exported_pipeline[:-1].transform(X_single)
#print(X_transformed)
X_transformed=pd.DataFrame(X_transformed)
pd.set_option('display.max_columns', None)
print(X_transformed)
print(X_transformed.shape)
print(type(X_transformed))
X_transformed=X_transformed.to_numpy()
print(type(X_transformed))
```

```
     0         1         2        3         4        5         6         7   \
0  0.0  0.000515  0.064838  0.15623  0.007124  0.06528  0.104452  0.063481

          8    9         10        11        12       13        14        15  \
0 -0.015353  0.0 -0.049261  0.000161  1.000508 -0.00322  0.000221 -0.004747

          16        17        18        19        20        21        22   23  \
0 -0.003683  0.130132  0.130099  0.002908  0.033837  0.032388  0.032388  0.0

     24   25
0  1.0  0.0
(1, 26)
```

Figure 83: Transformed Data Using Pipeline

```
decision node 0 : (X_transformed[0, 9] = 0.0) <= 7.200607180595398)
decision node 1 : (X_transformed[0, 8] = -0.015352903546804601) <= 1.157900333404541)
decision node 2 : (X_transformed[0, 9] = 0.0) <= 0.00044346097274683416)
decision node 3 : (X_transformed[0, 17] = 0.13013235436189768) <= 0.25504782795906067)
decision node 4 : (X_transformed[0, 11] = 0.00016107745227411888) > -2.2726287841796875)
decision node 10 : (X_transformed[0, 12] = 1.0005081344616336) > 0.8987369537353516)
decision node 242 : (X_transformed[0, 14] = 0.00022135663762750887) <= 19.47999095916748)
decision node 243 : (X_transformed[0, 12] = 1.0005081344616336) <= 1.0131675601005554)
decision node 244 : (X_transformed[0, 10] = -0.049261083743870296) <= 0.3620689660310745)
decision node 245 : (X_transformed[0, 8] = -0.015352903546804601) <= -0.00039093042141757905)
decision node 246 : (X_transformed[0, 10] = -0.049261083743870296) <= 0.002463054144755006)
decision node 247 : (X_transformed[0, 8] = -0.015352903546804601) <= -0.012545312289148569)
decision node 248 : (X_transformed[0, 10] = -0.049261083743870296) <= -0.04679802805185318)
decision node 249 : (X_transformed[0, 7] = 0.06348050668492289) <= 0.06350405141711235)
decision node 250 : (X_transformed[0, 10] = -0.049261083743870296) > -0.06650246307253838)
decision node 780 : (X_transformed[0, 7] = 0.06348050668492289) > 0.053358037024736404)
decision node 782 : (X_transformed[0, 16] = -0.0036833382159502806) <= -0.003438082290813327)
decision node 783 : (X_transformed[0, 11] = 0.00016107745227411888) > 0.00015328338486142457)
decision node 791 : (X_transformed[0, 10] = -0.049261083743870296) > -0.06157635524868965)
decision node 807 : (X_transformed[0, 15] = -0.004746703358386161) > -0.014418579172343016)
decision node 813 : (X_transformed[0, 16] = -0.0036833382159502806) > -0.010481876321136951)
decision node 931 : (X_transformed[0, 16] = -0.0036833382159502806) > -0.010474001057446003)
decision node 933 : (X_transformed[0, 12] = 1.0005081344616336) <= 1.0005816221237183)
decision node 934 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9588581025600433)
decision node 992 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9590879380702972)
```

Figure 84: Decision Tree Rules Part 1

```
decision node 1012 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9606534242630005)
decision node 1072 : (X_transformed[0, 14] = 0.00022135663762750887) > -0.006129589164629579)
decision node 1074 : (X_transformed[0, 12] = 1.0005081344616336) > 0.960730642080307)
decision node 1076 : (X_transformed[0, 7] = 0.06348050668492289) <= 0.06349680572748184)
decision node 1077 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.008414252195507288)
decision node 1093 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9612129330635071)
decision node 1095 : (X_transformed[0, 7] = 0.06348050668492289) > 0.054026348516345024)
decision node 1105 : (X_transformed[0, 15] = -0.004746703358386161) > -0.012428072281181812)
decision node 1107 : (X_transformed[0, 14] = 0.00022135663762750887) > -0.005832374328747392)
decision node 1121 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.008231109473854303)
decision node 1125 : (X_transformed[0, 16] = -0.0036833382159502806) > -0.009482852183282375)
decision node 1127 : (X_transformed[0, 15] = -0.004746703358386161) > -0.012135901488363743)
decision node 1131 : (X_transformed[0, 14] = 0.00022135663762750887) > -0.0057515420485287905)
decision node 1133 : (X_transformed[0, 7] = 0.06348050668492289) > 0.05410422757267952)
decision node 1135 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.008143654093146324)
decision node 1137 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.008124105632305145)
decision node 1145 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9638389945030212)
decision node 1175 : (X_transformed[0, 14] = 0.00022135663762750887) > -0.005521480226889253)
decision node 1181 : (X_transformed[0, 14] = 0.00022135663762750887) <= 0.00028726618620567024)
decision node 1182 : (X_transformed[0, 12] = 1.0005081344616336) > 0.9638936221599579)
decision node 1184 : (X_transformed[0, 14] = 0.00022135663762750887) > -0.005496608791872859)
decision node 1186 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.007944049779325724)
decision node 1188 : (X_transformed[0, 17] = 0.13013235436189768) > 0.12820017337799072)
decision node 1190 : (X_transformed[0, 14] = 0.00022135663762750887) > 0.00013803700858261436)
decision node 1196 : (X_transformed[0, 14] = 0.00022135663762750887) > 0.0001765878841979429)
decision node 1200 : (X_transformed[0, 16] = -0.0036833382159502806) > -0.0036957135889679193)
decision node 1206 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.003229681635275483)
decision node 1208 : (X_transformed[0, 13] = -0.0032204217126799305) > -0.0032235083635896444)
decision node 1214 : (X_transformed[0, 15] = -0.004746703358386161) <= -0.004727992927655578)
decision node 1215 : (X_transformed[0, 7] = 0.06348050668492289) <= 0.06348412856459618)
```

Figure 85: Decision Tree Rules Part 2

Appendix E TPOT Model Convergences

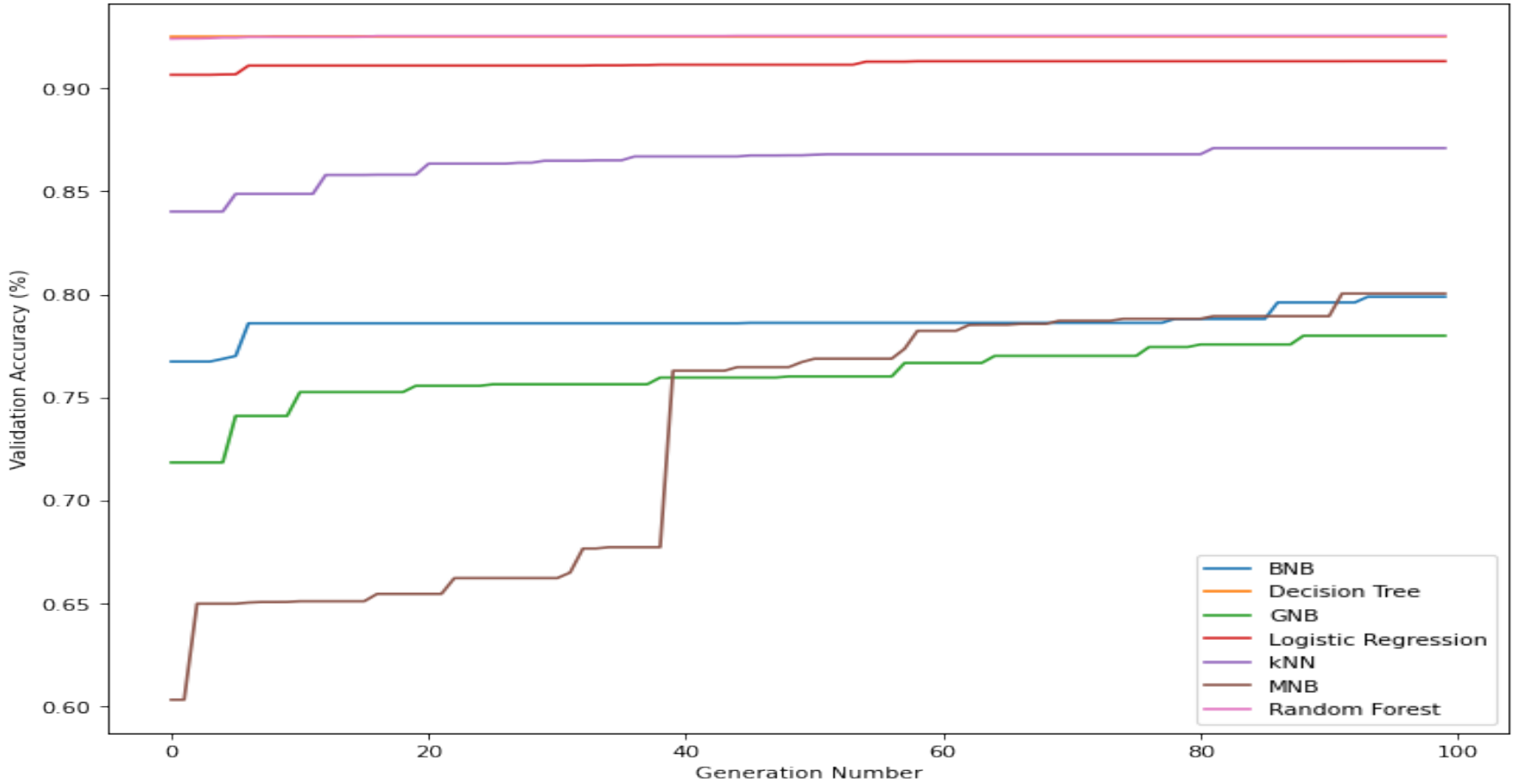


Figure 86: Expanded Model Training Convergence

Appendix F Misclassification Subset Descriptive Statistics

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMENT TEMPERATURE
count	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463
mean	2.399884	4.499567	568.829542	563.695339	563.695628	568.825435	563.693441	563.693746	628.7755	107.908945	0.00462	110.085484
std	0.006797	0.02549	11.817699	9.606949	9.590152	11.772501	9.557015	9.54022	12.95716	1.589987	0.27189	1.587145
min	2	3	4	5	6	7	8	9	14	15	0	17
25%	2.4	4.5	564.915	563.5385	563.5385	564.914	563.5395	563.5395	621.8845	107.758	0	109.974
50%	2.4	4.5	566.606	564.044	564.044	566.576	564.044	564.044	630.587	107.836	0	110.155
75%	2.4	4.5	568.88	564.7745	564.7745	568.905	564.7695	564.7695	636.214	108.059	0	110.177
max	2.4	4.5	617.653	566.4	566.4	617.696	566.394	566.394	650.722	108.282	16	110.295

Figure 87: Descriptive Statistics For Correct Turbine Trip W/O SCRAM Classification 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463	3463
mean	55.36446	136.2195	60.050057	61.077888	61.364596	59.321158	1134.518354	1134.514978	566.2673	2198.004482	650.11566	650.115945	-0.40945
std	15.89007	283.2588	108.43432	108.085059	107.708733	107.935823	23.764741	23.751778	9.772563	72.274191	11.312491	11.296684	6.883711
min	11.1965	0	3.23929	3.23635	4.30793	2.56344	26	27	28	29	31	32	-19.2639
25%	53.6622	0.001545	11.7241	11.7369	11.723	11.7135	1129	1128.995	564.499	2187.09	649.585	649.585	0
50%	54.0097	26.5167	22.9258	24.9839	25.6042	21.8508	1139.67	1139.67	564.996	2230.45	652.409	652.409	0
75%	57.84245	74.50675	38.84215	39.1878	40.85405	38.64025	1142.69	1142.69	566.758	2236.69	652.811	652.811	0.000003
max	93.9995	2466.26	663.292	663.445	654.684	654.294	1165.18	1165.21	588.547	2272.2	655.085	655.085	248.789

Figure 88: Descriptive Statistics For Correct Turbine Trip W/O SCRAM Classification 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAINMENT PRESSURE	CONTAINMENT TEMPERATURE
count	3.31E+02	331	331	331	331	331	331	331	331	331	331	331
mean	2.40E+00	4.5	564.631586	563.908625	563.908625	564.629526	563.907976	563.907991	629.9666	108.0346	0	110.197583
std	4.45E-16	0	0.211468	0.214358	0.214358	0.211136	0.21417	0.214169	2.669005	0.098041	0	0.03561
min	2.40E+00	4.5	564.36	563.632	563.632	564.358	563.631	563.631	626.519	107.913	0	110.159
25%	2.40E+00	4.5	564.362	563.635	563.635	564.361	563.635	563.635	627.172	107.972	0	110.175
50%	2.40E+00	4.5	564.736	564.015	564.015	564.734	564.014	564.014	630.681	107.983	0	110.175
75%	2.40E+00	4.5	564.745	564.024	564.024	564.7435	564.023	564.023	631.002	108.145	0	110.242
max	2.40E+00	4.5	564.915	564.198	564.198	564.914	564.197	564.197	636.164	108.214	0	110.249

Figure 89: Descriptive Statistics For Correct Turbine Trip W/O SCRAM Misclassified as Feedwater Pump Trips 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	331	331	331	331	331	331	331	331	331	331	331	331	331
mean	54.36158	25.27403	11.429719	11.429965	11.408767	11.408209	1139.948066	1139.947946	564.2694	2233.721	652.619665	652.619665	0
std	0.261932	0.618486	0.219735	0.219873	0.224273	0.224066	1.993696	1.99378	0.212739	2.37301	0.152977	0.152977	0
min	54.001	24.0727	11.0284	11.0288	10.9994	10.9993	1137.38	1137.38	563.995	2231.19	652.457	652.457	0
25%	54.1723	24.83625	11.24105	11.2416	11.21575	11.2154	1137.41	1137.41	563.998	2231.32	652.465	652.465	0
50%	54.2924	25.1988	11.428	11.428	11.4008	11.4001	1140.94	1140.94	564.375	2232.55	652.544	652.544	0
75%	54.53825	25.44825	11.595	11.59515	11.57675	11.57645	1141.02	1141.02	564.384	2236.595	652.805	652.805	0
max	55.1014	26.8595	11.8981	11.8982	11.8907	11.8889	1142.66	1142.66	564.556	2236.93	652.827	652.827	0

Figure 90: Descriptive Statistics For Correct Turbine Trip W/O SCRAM Misclassified as Feedwater Pump Trips 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMENT TEMPERATURE
count	883	883	883	883	883	883	883	883	883	883	883	883
mean	2.4	4.5	564.684535	563.963523	563.963523	564.683199	563.963151	563.963176	631.4075	108.044864	0	110.192612
std	0	0	0.21909	0.222348	0.222348	0.218972	0.222299	0.222301	2.933058	0.101418	0	0.035672
min	2.4	4.5	564.36	563.632	563.632	564.358	563.632	563.632	626.524	107.913	0	110.158
25%	2.4	4.5	564.376	563.652	563.652	564.375	563.652	563.652	629.865	107.975	0	110.166
50%	2.4	4.5	564.76	564.043	564.043	564.76	564.044	564.044	630.692	108.013	0	110.175
75%	2.4	4.5	564.868	564.148	564.148	564.866	564.1475	564.1475	633.3085	108.143	0	110.241
max	2.4	4.5	564.918	564.202	564.202	564.918	564.202	564.202	636.274	108.228	0	110.249

Figure 91: Descriptive Statistics Turbine Trip W/O SCRAM Misclassified as Electrical Load Rejection 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	883	883	883	883	883	883	883	883	883	883	883	883	883
mean	54.22098	25.80146	11.563684	11.564058	11.548913	11.548663	1140.467848	1140.467803	564.3236	2234.749841	652.68605	652.686051	0
std	0.266821	0.827682	0.234786	0.234762	0.24233	0.242378	2.073212	2.073177	0.220677	2.156003	0.13902	0.13902	0
min	53.9999	23.8872	11.0239	11.0243	10.995	10.9945	1137.38	1137.38	563.996	2231.19	652.457	652.457	0
25%	54.0035	25.10005	11.32905	11.32915	11.3059	11.30565	1137.56	1137.56	564.014	2232.66	652.551	652.551	0
50%	54.1322	25.4787	11.6156	11.6158	11.5981	11.598	1141.23	1141.23	564.402	2235.15	652.712	652.712	0
75%	54.3566	26.5421	11.72625	11.72675	11.71595	11.71565	1142.18	1142.18	564.5075	2236.73	652.814	652.814	0
max	55.0719	26.8916	11.901	11.9018	11.8958	11.8963	1142.7	1142.7	564.56	2236.94	652.827	652.827	0

Figure 92: Descriptive Statistics Turbine Trip W/O SCRAM Misclassified as Electrical Load Rejection 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMENT TEMPERATURE
count	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797
mean	2.399857	4.499464	569.545156	563.6348	563.635158	569.539337	563.632059	563.632432	628.1932	107.873661	0.00572	110.061203
std	0.007563	0.028363	12.793543	10.695557	10.676877	12.741256	10.6401	10.621425	14.13069	1.766417	0.302534	1.765043
min	2	3	4	5	6	7	8	9	14	15	0	17
25%	2.4	4.5	566.111	563.486	563.486	566.096	563.489	563.489	620.177	107.751	0	109.953
50%	2.4	4.5	567.741	564.018	564.018	567.736	564.016	564.016	629.883	107.805	0	110.091
75%	2.4	4.5	568.984	565.259	565.259	569.008	565.255	565.255	636.271	108.058	0	110.176
max	2.4	4.5	617.448	566.275	566.275	617.665	566.255	566.255	650.379	108.282	16	110.295

Figure 93: Descriptive Statistics for Correct Electrical Load Rejection 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797	2797
mean	53.77906	162.777	66.297688	67.543239	67.865301	65.380211	1133.313672	1133.309271	566.5957	2189.004448	649.4863	649.486661	0.187756
std	15.7703	298.9336	108.717415	108.220939	107.78376	108.241424	26.429862	26.415253	10.76225	78.32602	12.521814	12.50415	0.706326
min	11.1421	0	4.6608	4.66684	2.89355	1.1257	26	27	28	29	31	32	0
25%	50.4755	0.000601	11.8991	14.2123	12.388	11.889	1128.42	1128.42	564.669	2174.36	648.748	648.748	0
50%	54.0393	26.5343	26.3184	28.8807	29.3546	25.3229	1137.56	1137.56	566.484	2220.45	651.762	651.762	0
75%	59.8175	132.321	53.5267	53.5324	54.7781	52.9636	1143.41	1143.41	567.133	2234.93	652.698	652.698	0.050259
max	93.7951	2473.01	663.444	663.584	653.4	652.964	1162.1	1162.13	588.428	2272.49	655.104	655.104	33

Figure 94: Descriptive Statistics for Correct Electrical Load Rejection 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMEN T TEMPERATURE
count	8.99E+02	899	899	899	899	899	899	899	899	899	899	899
mean	2.40E+00	4.5	564.693268	563.969804	563.969804	564.692067	563.969487	563.969508	631.7141	108.053892	0	110.193303
std	4.44E-16	0	0.260791	0.232335	0.232335	0.260363	0.232227	0.232221	3.093963	0.102015	0	0.036194
min	2.40E+00	4.5	564.36	563.632	563.632	564.358	563.631	563.631	626.519	107.832	0	110.15
25%	2.40E+00	4.5	564.376	563.652	563.652	564.375	563.652	563.652	629.875	107.982	0	110.166
50%	2.40E+00	4.5	564.76	564.043	564.043	564.761	564.044	564.044	630.675	108.014	0	110.175
75%	2.40E+00	4.5	564.877	564.156	564.156	564.8745	564.155	564.155	633.5695	108.147	0	110.243
max	2.40E+00	4.5	568.715	565.484	565.484	568.694	565.474	565.474	636.275	108.228	0	110.249

Figure 95: Descriptive Statistics for Electrical Load Rejection Misclassified as Feedwater Pump Trip 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULA TED AVERAGE TEMPERA TURE	PRESSURIZER PRESSURE	PRESSURI ZER WATER TEMPERA TURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	899	899	899	899	899	899	899	899	899	899	899	899	899
mean	54.18784	25.98728	11.657567	11.657886	11.644958	11.642688	1140.523504	1140.523437	564.3312	2234.954483	652.69923	652.699231	0
std	0.653702	0.80678	1.529976	1.529825	1.555302	1.496634	2.141594	2.141624	0.24326	2.129519	0.137308	0.137308	0
min	54.0003	24.1304	11.0282	11.0282	10.999	10.999	1137.38	1137.38	563.995	2222.43	651.89	651.89	0
25%	54.0034	25.1972	11.38765	11.38775	11.36085	11.36065	1137.56	1137.56	564.014	2232.845	652.5635	652.5635	0
50%	54.0088	26.5168	11.7019	11.7024	11.7062	11.706	1141.23	1141.23	564.402	2235.24	652.718	652.718	0
75%	54.29485	26.5607	11.7343	11.7348	11.7223	11.72195	1142.25	1142.25	564.516	2236.73	652.814	652.814	0
max	72.3785	26.8956	56.9447	56.9401	57.6658	55.8854	1150.56	1150.56	567.092	2236.93	652.827	652.827	0

Figure 96: Descriptive Statistics for Electrical Load Rejection Misclassified as Feedwater Pump Trip 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAINMENT PRESSURE	CONTAINMENT TEMPERATURE
count	7.14E+02	714	714	714	714	714	714	714	714	714	714	714
mean	2.40E+00	4.5	567.255714	564.653423	564.653422	567.268951	564.65636	564.656371	626.120284	108.151157	0	110.22527
std	4.44E-16	0	3.226798	0.849704	0.849702	3.241345	0.851441	0.851438	5.058801	0.142956	0	0.04648
min	2.40E+00	4.5	564.36	562.098	562.098	564.358	562.099	562.099	613.964	107.737	0	109.915
25%	2.40E+00	4.5	564.736	564.012	564.012	564.73325	564.011	564.011	621.9575	107.98625	0	110.175
50%	2.40E+00	4.5	568.812	565.3665	565.3665	568.838	565.372	565.372	622.4545	108.153	0	110.249
75%	2.40E+00	4.5	568.89375	565.439	565.439	568.91975	565.444	565.444	630.778	108.274	0	110.25
max	2.40E+00	4.5	594.114	565.599	565.599	594.17	565.598	565.598	643.113	108.283	0	110.295

Figure 97: Descriptive Statistics for Electrical Load Rejection Misclassified as Turbine Trip W/O SCRAM 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	714	714	714	714	714	714	714	714	714	714	714	714	714
mean	54.384544	99.3978	38.031885	38.045198	38.724268	37.736171	1141.28105	1141.279622	565.958615	2231.585574	652.481	652.480557	0.01145
std	2.861543	127.373	39.562648	39.556872	39.995793	39.4953	3.720954	3.719822	1.8295	10.833877	0.71491	0.714913	0.10014
min	25.3207	0	11.03	11.03	11.001	11.0009	1113.47	1113.47	563.995	2064.99	641.388	641.388	0
25%	54.0359	25.1134	11.5425	11.54265	11.519325	11.51815	1137.6925	1137.695	564.374	2230.65	652.421	652.42125	0
50%	54.0443	131.865	53.48245	53.4833	54.7289	52.92015	1142.695	1142.695	567.0955	2232.345	652.531	652.531	0
75%	54.491575	132.421	53.662075	53.66265	54.908975	53.099225	1143.3175	1143.31	567.174	2234.8875	652.695	652.69475	0
max	82.8546	1180.98	457.925	457.903	458.667	457.545	1150.25	1150.25	578.718	2246.51	653.442	653.442	1.1977

Figure 98: Descriptive Statistics for Electrical Load Rejection Misclassified as Turbine Trip W/O SCRAM 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMENT TEMPERATURE
count	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290
mean	2.399878	4.499544	568.410268	564.228057	564.22836	568.388957	564.211537	564.211858	626.5618	107.920579	0.004863	110.086346
std	0.006974	0.026151	12.117809	9.833602	9.816312	12.057294	9.780805	9.763516	12.99108	1.633761	0.278947	1.628763
min	2	3	4	5	6	7	8	9	14	15	0	17
25%	2.4	4.5	564.915	563.651	563.651	564.914	563.651	563.651	619.6595	107.756	0	109.97
50%	2.4	4.5	567.481	564.58	564.58	567.466	564.5765	564.5765	627.33	107.829	0	110.15
75%	2.4	4.5	568.40775	565.273	565.273	568.37075	565.2315	565.2315	630.6745	108.121	0	110.25
max	2.4	4.5	619.802	569.18	569.18	619.975	569.141	569.141	651.335	108.283	16	110.295

Figure 99: Descriptive Statistics for Correct Feedwater Pump Trip 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290	3290
mean	12.91062	10.12443	65.742552	69.65015	69.907142	64.942509	1125.357356	1125.35369	566.3164	2191.073152	649.65419	649.654495	17.590303
std	21.84783	82.22403	127.125899	126.958452	125.887147	126.216334	29.254003	29.240108	10.06189	73.166482	11.568587	11.552329	135.570604
min	0	0	11.1122	11.1123	11.087	11.0868	26	27	28	29	31	32	-19.3392
25%	0	0	12.81765	18.0638	18.909525	12.154825	1120.9775	1120.9775	564.556	2162.99	648.00075	648.00075	0
50%	0	0	49.343	54.55845	55.3619	48.72735	1134.59	1134.585	566.107	2215.455	651.4375	651.4375	0
75%	23.7605	0.107691	57.70165	62.7868	64.311	57.818575	1137.56	1137.56	566.712	2236.295	652.7855	652.7855	0.000022
max	55.0651	2011.51	1244.1	1244.14	1249.02	1248.92	1172.34	1172.32	591.966	2279.6	655.555	655.555	1549.1

Figure 100: Descriptive Statistics for Correct Feedwater Pump Trip 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAINMENT PRESSURE	CONTAINMENT TEMPERATURE
count	883	883	883	883	883	883	883	883	883	883	883	883
mean	2.4	4.5	564.684535	563.963523	563.963523	564.683199	563.963151	563.963176	631.40745	108.044864	0	110.192612
std	0	0	0.21909	0.222348	0.222348	0.218972	0.222299	0.222301	2.933058	0.101418	0	0.035672
min	2.4	4.5	564.36	563.632	563.632	564.358	563.632	563.632	626.524	107.913	0	110.158
25%	2.4	4.5	564.376	563.652	563.652	564.375	563.652	563.652	629.865	107.975	0	110.166
50%	2.4	4.5	564.76	564.043	564.043	564.76	564.044	564.044	630.692	108.013	0	110.175
75%	2.4	4.5	564.868	564.148	564.148	564.866	564.1475	564.1475	633.3085	108.143	0	110.241
max	2.4	4.5	564.918	564.202	564.202	564.918	564.202	564.202	636.274	108.228	0	110.249

Figure 101: Descriptive Statistics for Feedwater Pump Trip Misclassified as Load Rejection 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	883	883	883	883	883	883	883	883	883	883	883	883	883
mean	54.22098	25.80146	11.563684	11.564058	11.548913	11.548663	1140.467848	1140.467803	564.323596	2234.749841	652.68605	652.686051	0
std	0.266821	0.827682	0.234786	0.234762	0.24233	0.242378	2.073212	2.073177	0.220677	2.156003	0.13902	0.13902	0
min	53.9999	23.8872	11.0239	11.0243	10.995	10.9945	1137.38	1137.38	563.996	2231.19	652.457	652.457	0
25%	54.0035	25.10005	11.32905	11.32915	11.3059	11.30565	1137.56	1137.56	564.014	2232.66	652.551	652.551	0
50%	54.1322	25.4787	11.6156	11.6158	11.5981	11.598	1141.23	1141.23	564.402	2235.15	652.712	652.712	0
75%	54.3566	26.5421	11.72625	11.72675	11.71595	11.71565	1142.18	1142.18	564.5075	2236.73	652.814	652.814	0
max	55.0719	26.8916	11.901	11.9018	11.8958	11.8963	1142.7	1142.7	564.56	2236.94	652.827	652.827	0

Figure 102: Descriptive Statistics for Feedwater Pump Trip Misclassified as Load Rejection 2 of 2

	RCS LVL LOOP 1 WR	RCS LVL LOOP 1 NR	HOT LEG 1 TEMPERATURE	COLD LEG 1A "TEMPERATURE	COLD LEG 1B TEMPERATURE	HOT LEG 2 TEMPERATURE	COLD LEG 2A TEMPERATURE	COLD LEG 2B TEMPERATURE	PZR SURGE LINE TEMP	PORV DISCH PZR TEMPERATURE	CONTAIN MENT PRESSURE	CONTAINMENT TEMPERATURE
count	883	883	883	883	883	883	883	883	883	883	883	883
mean	2.4	4.5	564.684535	563.963523	563.963523	564.683199	563.963151	563.963176	631.4075	108.044864	0	110.192612
std	0	0	0.21909	0.222348	0.222348	0.218972	0.222299	0.222301	2.933058	0.101418	0	0.035672
min	2.4	4.5	564.36	563.632	563.632	564.358	563.632	563.632	626.524	107.913	0	110.158
25%	2.4	4.5	564.376	563.652	563.652	564.375	563.652	563.652	629.865	107.975	0	110.166
50%	2.4	4.5	564.76	564.043	564.043	564.76	564.044	564.044	630.692	108.013	0	110.175
75%	2.4	4.5	564.868	564.148	564.148	564.866	564.1475	564.1475	633.3085	108.143	0	110.241
max	2.4	4.5	564.918	564.202	564.202	564.918	564.202	564.202	636.274	108.228	0	110.249

Figure 103: Descriptive Statistics for Feedwater Pump Trip Misclassified as Turbine Trips W/O SCRAM 1 of 2

	SG-1 NR LEVEL	SG-2 NR LEVEL	FW FLOW TO SG-2	MS FLOW FROM SG-1 LINE-1A	MS FLOW FROM SG-1 LINE-1B	MS FLOW FROM SG-2 LINE-2A	MS FLOW FROM SG-2 LINE-2B	SG-2 PRESSURE	CALCULATED AVERAGE TEMPERATURE	PRESSURIZER PRESSURE	PRESSURIZER WATER TEMPERATURE	PRESSURIZER STEAM TEMPERATURE	GENERATOR POWER
count	883	883	883	883	883	883	883	883	883	883	883	883	883
mean	54.22098	25.80146	11.563684	11.564058	11.548913	11.548663	1140.467848	1140.467803	564.3236	2234.749841	652.68605	652.686051	0
std	0.266821	0.827682	0.234786	0.234762	0.24233	0.242378	2.073212	2.073177	0.220677	2.156003	0.13902	0.13902	0
min	53.9999	23.8872	11.0239	11.0243	10.995	10.9945	1137.38	1137.38	563.996	2231.19	652.457	652.457	0
25%	54.0035	25.10005	11.32905	11.32915	11.3059	11.30565	1137.56	1137.56	564.014	2232.66	652.551	652.551	0
50%	54.1322	25.4787	11.6156	11.6158	11.5981	11.598	1141.23	1141.23	564.402	2235.15	652.712	652.712	0
75%	54.3566	26.5421	11.72625	11.72675	11.71595	11.71565	1142.18	1142.18	564.5075	2236.73	652.814	652.814	0
max	55.0719	26.8916	11.901	11.9018	11.8958	11.8963	1142.7	1142.7	564.56	2236.94	652.827	652.827	0

Figure 104: Descriptive Statistics for Feedwater Pump Trip Misclassified as Turbine Trips W/O SCRAM 2 of 2

Appendix G Plots from Autoencoder

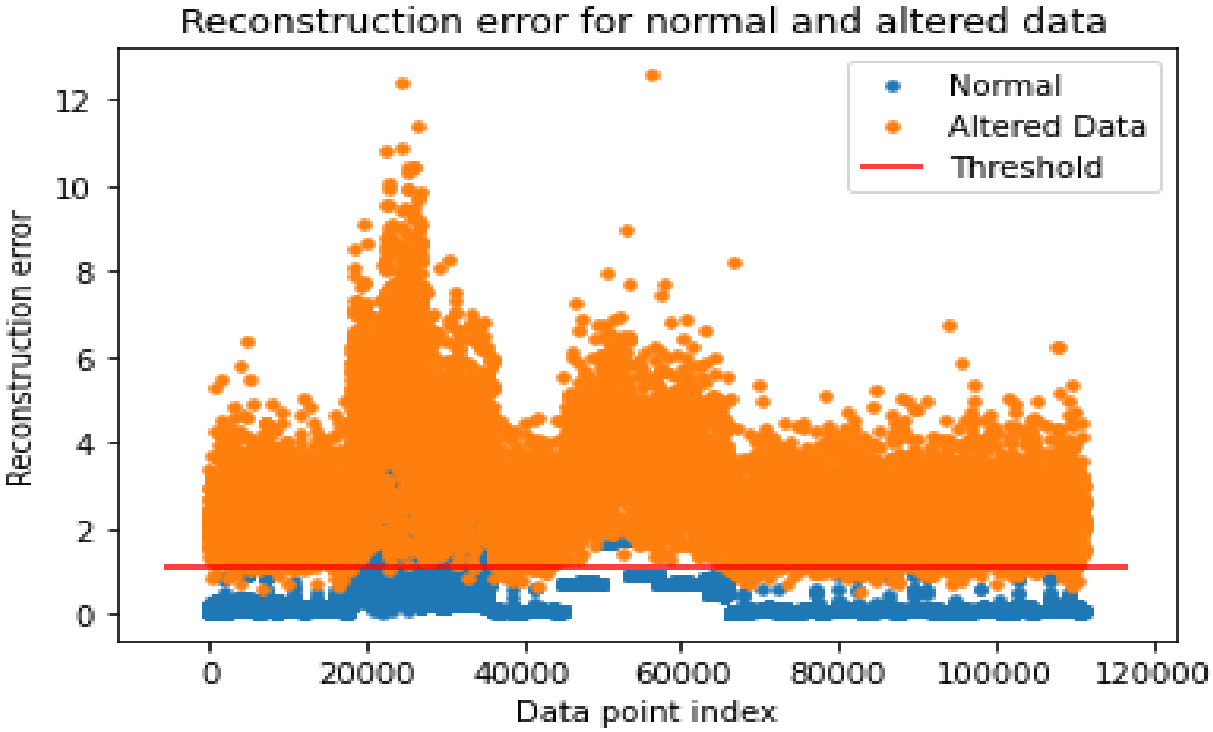


Figure 105: Reconstruction Plot for Autoencoder at 1.5 SD Noise

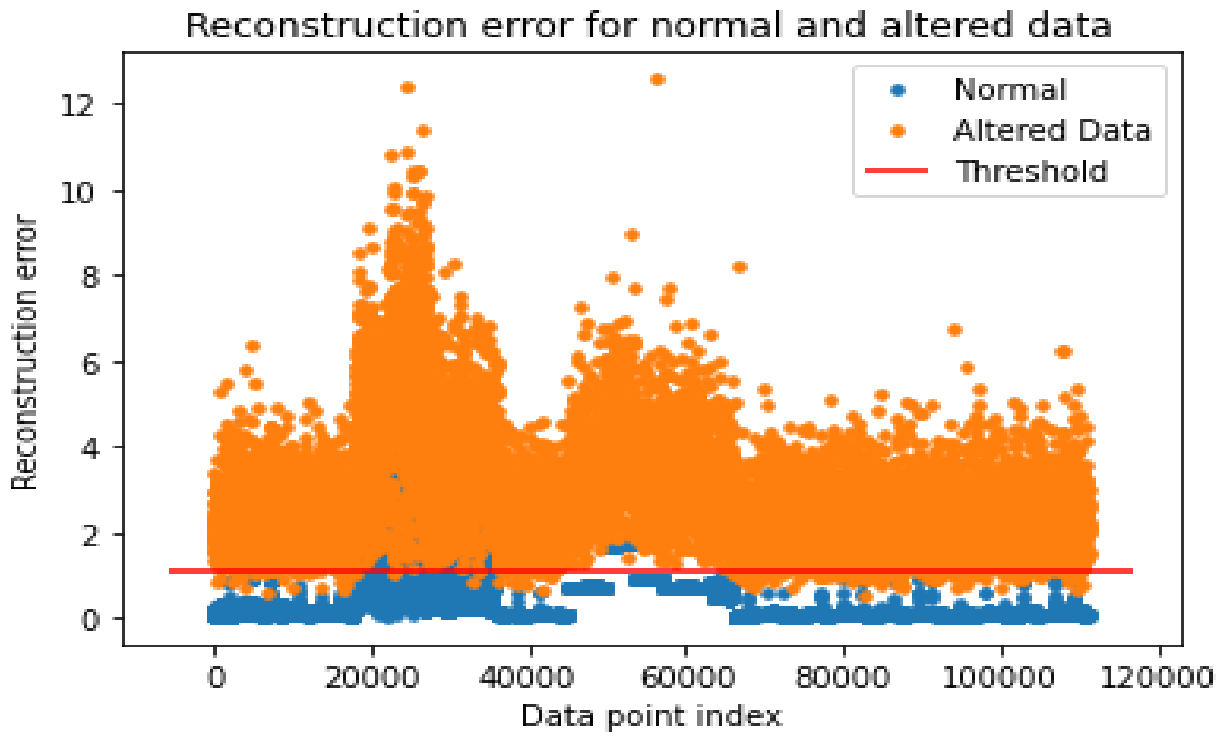


Figure 106: Reconstruction Plot for Autoencoder at 1.0 SD Noise

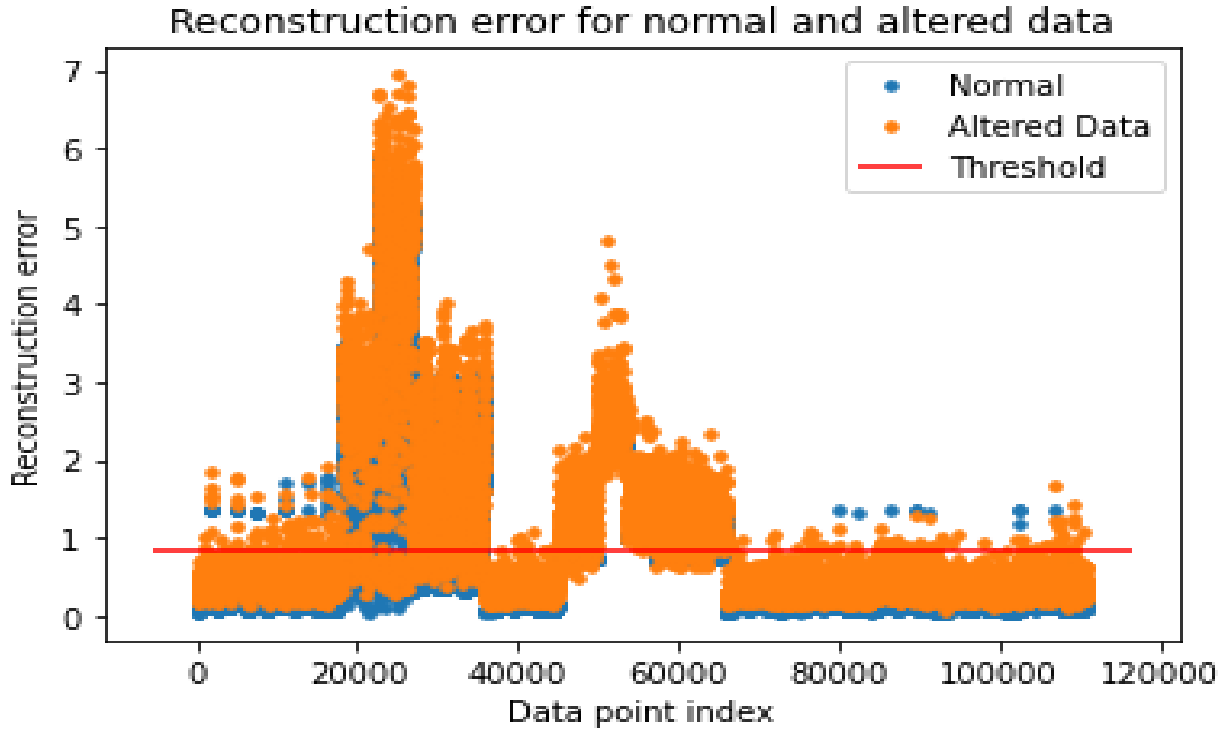


Figure 107: Reconstruction Plot for Autoencoder at 0.5 SD Noise