Summer 7-5-2017

A NOVEL APPLICATION OF MACHINE LEARNING METHODS TO MODEL MICROCONTROLLER UPSET DUE TO INTENTIONAL ELECTROMAGNETIC INTERFERENCE

Rusmir Bilalic
University of New Mexico - Main Campus

Rusmir Bilalic

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Dr. Edl Schamiloglu, Chairperson

Dr. Christos Christodoulou

Dr. Manel Martinez-Ramon

Dr. Trilce Estrada

Dr. Timothy Clarke

Dr. Sameer Hemmady

A NOVEL APPLICATION OF MACHINE LEARNING METHODS TO MODEL MICROCONTROLLER UPSET DUE TO INTENTIONAL ELECTROMAGNETIC INTERFERENCE

by

RUSMIR BILALIC

B.S Electrical Engineering, Northeastern University,
Boston, MA, 2005

M.S. Electrical Engineering, Northeastern University,
Boston, MA, 2009

M.S. Strategic Intelligence, National Intelligence University,
Washington, DC, 2014

Ph.D. Engineering, University of New Mexico
Albuquerque, NM, 2017

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

July 2017

DEDICATION

To my family.

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to my committee chair and advisor, Prof. Edl Schamiloglu for giving me the freedom and flexibility to pursue a topic of my interest. His guidance, mentorship, and support during stressful qualifying exam periods as well as his encouragement and support during the experimentation and analysis phases of this project enabled me to complete my degree requirements on a very short timeline.

Next, I gratefully acknowledge the assistance and support of my committee members: Dean Christos Christodoulou, Prof. Manel Martinez-Ramon, Prof. Trilce Estrada, Dr. Tim Clarke, and Prof. Sameer Hemmady. Their invaluable feedback and encouragement enabled me to pursue this topic with confidence. I am especially indebted to Dr. Clarke and company for allowing me to use AFRL facilities and equipment 24/7/365. Without his generous support, none of this would have been possible. Big thanks to Prof. Sal Portillo for his positive encouragement and for being my sounding board for ideas with very candid feedback -- he never doubted that any of this would happen. “Embryonic Dr. Rus” he would say.

Additionally, I gratefully acknowledge the Air Force Institute of Technology, big Air Force, and my previous commanders for selecting me for this program and the very generous financial support during my studies. Also thanks to Mr. Daniel Clepper, Lt Col Garcia, Lt Col Kleinschmidt, and the UNM ROTC Det 510 crew for the administrative support during my time at UNM.

I would also like to extend my sincere gratitude to my dear friend, Mr. Daniel Guillette. This task would have been much more complex had it not been for his technical expertise and motivational support. I would also like to take this opportunity and gratefully acknowledge the support by Dr. Julie Lawrance, Mr. Matt Landavazo, Mr. Hugh Pohle, and other future colleagues within AFRL.

Next, enormous thanks to my family who sacrificed everything so that my brother and I can have opportunities that they never did. Finally, I am eternally thankful to my wife Jasna, who endures the constant moves and uncertainty that comes with our lifestyle. Her love and support, and her willingness to stand by me no matter what means the world to me. I could go on and on for pages, and I would still fail to acknowledge everyone whose support contributed to my success. To everyone that I failed to mention, a big thank you!

A NOVEL APPLICATION OF MACHINE LEARNING METHODS TO MODEL MICROCONTROLLER UPSET DUE TO INTENTIONAL ELECTROMAGNETIC INTERFERENCE

by

RUSMIR BILALIC

B.S Electrical Engineering, Northeastern University,
Boston, MA, 2005

M.S. Electrical Engineering, Northeastern University,
Boston, MA, 2009

M.S. Strategic Intelligence, National Intelligence University,
Washington, DC, 2014

Ph.D. Engineering, University of New Mexico
Albuquerque, NM, 2017

ABSTRACT

A novel application of support vector machines (SVMs), artificial neural networks (ANNs), and Gaussian processes (GPs) for machine learning (GPML) to model microcontroller unit (MCU) upset due to intentional electromagnetic interference (IEMI) is presented. In this approach, an MCU performs a counting operation (0-7) while electromagnetic interference in the form of a radio frequency (RF) pulse is direct-injected into the MCU clock line. Injection times with respect to the clock signal are the clock low, clock rising edge, clock high, and the clock falling edge periods in the clock window during which the MCU is performing initialization and executing the counting procedure. The intent is to cause disruption in the counting operation and model the probability of effect (PoE) using machine learning tools. Five experiments were executed as part of this research, each of which contained a set of 38,300 training points and 38,300 test points, for a total of 383,000 total points with the following experiment variables: injection times with respect to the clock signal, injected RF power, injected RF pulse width, and injected RF frequency. For the 191,500 training points, the average training error was 12.47%, while for the 191,500 test points the average test error was 14.85%, meaning that on average, the machine was able to predict MCU upset with an 85.15% accuracy. Leaving out the results for the worst-performing model (SVM with a linear kernel), the test prediction accuracy for the remaining machines is almost 89%. All three machine learning methods (ANNs, SVMs, and GPML) showed excellent and consistent results in their ability to model and predict the PoE on an MCU due to IEMI. The GP approach performed best during training with a 7.43% average training error, while the ANN technique was most accurate during the test with a 10.80% error.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction and Research Contribution

Industrialized nations today depend on the fast flow of information as well as information processing systems that make that flow possible. A vast majority of these information processing systems and infrastructure is not hardened against electromagnetic pulse (EMP) and EMP-producing weapon technology [1]. In addition to unintentional sources of electromagnetic (EM) radiation that can adversely affect critical electronic infrastructure, advanced militaries are experimenting with the use of high-power microwave (HPM) weapon systems. These systems are referred to as directed energy weapons (DEW), and their EM radiation can be used to upset or damage electronic systems. DEWs can be classified into three categories: lasers, microwave or radio frequency (RF) energy weapons also known as HPM weapons, and charged particle beam weapons. HPM weapons do not face the same propagation issues as lasers and charged particle beam weapons. Particle beams and lasers have difficulty propagating in the atmosphere, and electron beams cannot propagate in space, whereas HPM weapons propagate through a diffraction-limited manner and can accommodate some lack of precision, in contrast to the other types of DEWs [1].

The level of technological sophistication that is required to cause electronic system malfunction is not only available to modern militaries but also to non-state actors who were previously considered technologically backward. Fig. 1 illustrates the threat level vs. required technical knowledge to execute an HPM attack.

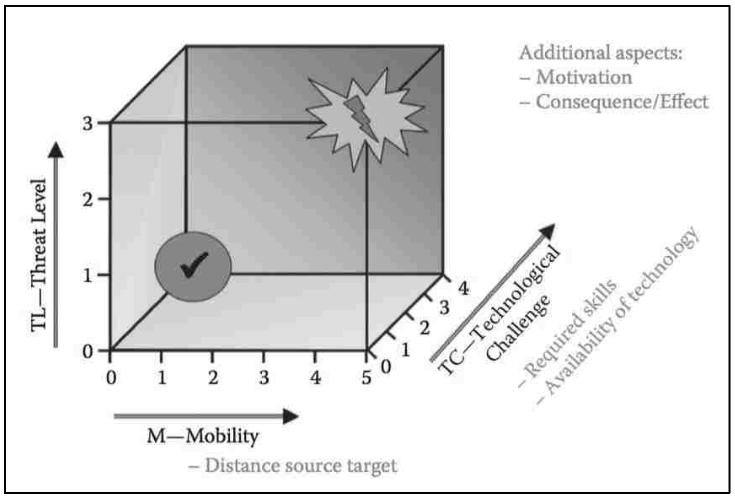


Figure 1: Threat level vs. technical expertise [1].

As Fig. 1 shows, this technology is becoming more accessible at a reasonable cost. Potential EMP targets range from small systems, such as automobiles, to highly complex systems, such as aircraft, energy and power delivery infrastructure, as well as systems used in the banking and financial sectors [1]. Indeed, there have been several instances of HPEM weapons use against civilian and military infrastructure [2].

Electromagnetic interference (EMI), as well as intentional electromagnetic interference (IEMI), are two important concepts related to the development and protection of critical electronic equipment for civilian and military use. IEMI/EMI can render civilian and military systems inoperable and can present significant obstacles to mission accomplishment, whether in the context of military operations or civilian use in hospitals, airports, or industry.

Counter-electronics High Power Microwave Advanced Missile Project (CHAMP) is a joint concept technology demonstration championed by the Air Force Research Laboratory Directed Energy Directorate (AFRH/RD) at Kirtland Air Force Base in Albuquerque, New Mexico. Its purpose is to demonstrate a military capability of an airlaunched DEW platform capable of incapacitating or damaging electronic systems. Test demonstrations were conducted in which the platform was successful in disrupting the normal operation of electronic systems in a test environment. In fact, a recent National Academy study, the CHAMP platform was described as having a DEW capability with a defined concept of operations (CONOPS), meaning there exist military plans to utilize such DEWs. These CONOPS entail non-kinetic counter-electronics (NKCE) attacks of value in operations that fall short of kinetic engagement, in which the re-establishment of normal target operations might be of operational interest [1]. In other words, the EM energy released by a CHAMP-like platform may have the objective to temporarily disrupt, but not destroy, the operation of electronic systems. In such cases, the operation is restored shortly after the attack.

After EM energy penetrates electronic systems enclosures, the predictable susceptibility of small-scale semiconductor devices becomes critical for DEW utility assessment [1]. The type of malfunction that this EM energy causes to semiconductor devices as well as predictive modeling of such malfunctions is not well understood, and it is the subject of this research.

Before research into effects of electronic systems due to unwanted EM radiation is presented, it is important to define some concepts that relate to IEMI/EMI. In a recent special issue paper, Radasky et al. describe high-power electromagnetics (HPEM) and IEMI as a new threat that can compromise the modern technology on which western economies and the military rely [2]. The term HPEM encompasses a set of transient EM pulses in which the peak fields can be of such intensity to cause an undesired operation or malfunction of electronic systems. Phenomena that can produce such EM transients include lighting strikes, electrostatic discharge (ESD), EMP created by nuclear bursts or other weapons, as well as EM fields generated by radar equipment [2], and directed energy weapons such as CHAMP. The phenomenon where the EM interference is intended to cause disruption to electronic systems is referred to as IEMI, and it is the primary focus of this research.

Several attributes can be used to describe and classify IEMI/IEME. The first is related to the EM field that is impinging on an electronic system. Fig. 2 shows the spectral classification of such an electric field as a function of frequency. The second attribute is related to the source that is producing the undesired IEMI. The third attribute that is used to describe and classify IEMI is the effects of the IEMI on the electronic systems [3]. Modeling and prediction of IEMI effects are the primary focus of this research.

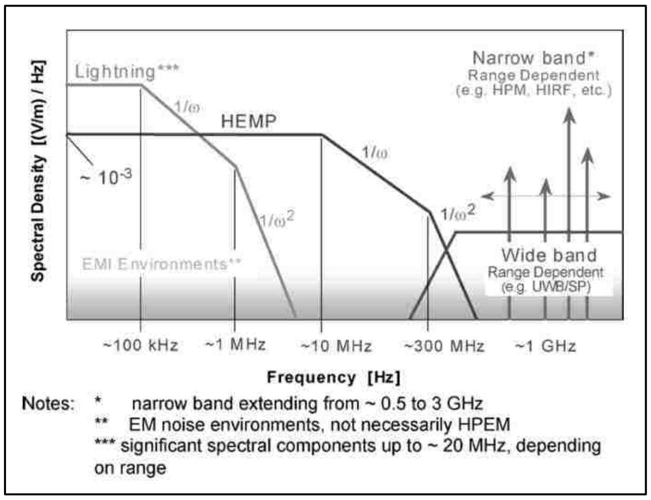
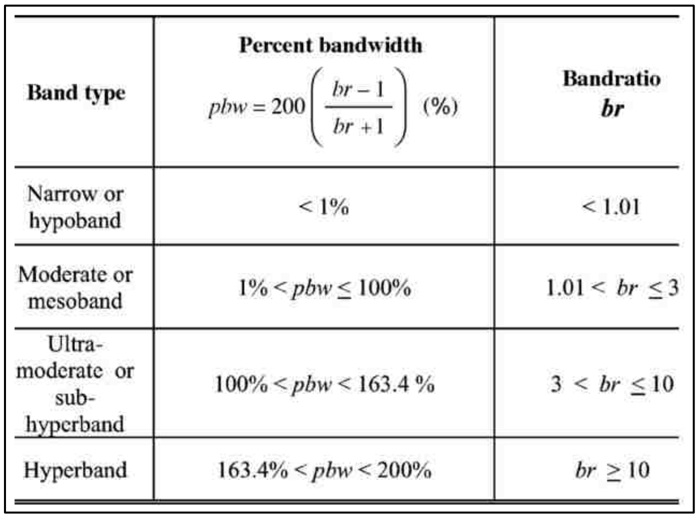


Figure 2: The spectral distribution of different types of electromagnetic environments [3].

Fig. 2 shows the EM spectrum of natural lightning, high-altitude EMP (HEMP), ultrawideband (UWB) EM, and HPM environments in the narrowband. Naturally occurring EMP phenomena, such as lightning, are not of primary focus. The two IEMI environments mentioned above can be characterized into four categories based on the frequency content of their spectral densities as follows: narrowband, moderate band, ultramoderate band, and hyperband.

The parameter that is used to characterize IEME is the band ratio (br), and it is defined as $br = \frac{f_h}{f_l}$ where f_h and f_l are denoted as the high and low frequency of operation, respectively. Although different agencies, namely the Federal Communications Commission (FCC) and the Defense Advanced Research Projects Agency (DARPA), utilize different classifications, the bandwidths that are sufficiently useful for an introductory treatment are summarized in Table 1 [3].

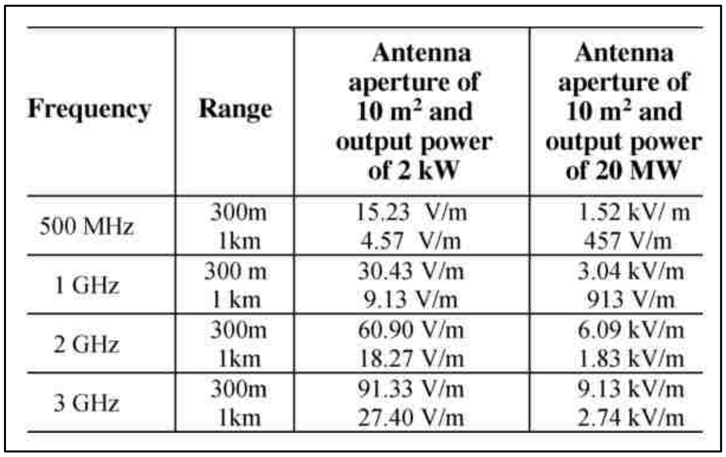
Table 1: IEME classification based on bandwidth [3].



Band type	**Percent bandwidth** $pbw = 200\left(\frac{br-1}{br+1}\right)$ (%)	**Bandratio** ***br***
Narrow or hypoband	$< 1\%$	< 1.01
Moderate or mesoband	$1\% < pbw \leq 100\%$	$1.01 < br \leq 3$
Ultra-moderate or sub-hyperband	$100\% < pbw < 163.4\%$	$3 < br \leq 10$
Hyperband	$163.4\% < pbw < 200\%$	$br \geq 10$

Alternatively, IEMI environments can be characterized and classified based on the electric field strength that is produced by the HPEM source at a specified distance away from the source. Table 2 shows the radiated electric fields at certain distances as produced by relatively compact antennas using readily-available microwave sources.

Table 2: Radiated electric fields generated by HPM systems [3].



Frequency	Range	Antenna aperture of 10 m^2 and output power of 2 kW	Antenna aperture of 10 m^2 and output power of 20 MW
500 MHz	300m	15.23 V/m	1.52 kV/ m
	1km	4.57 V/m	457 V/m
1 GHz	300 m	30.43 V/m	3.04 kV/m
	1 km	9.13 V/m	913 V/m
2 GHz	300m	60.90 V/m	6.09 kV/m
	1km	18.27 V/m	1.83 kV/m
3 GHz	300m	91.33 V/m	9.13 kV/m
	1km	27.40 V/m	2.74 kV/m

The final category by which IEMI environments can be classified is the technological level of the sources that are used to generate HPM. These sources can be broadly classified as low-tech generator systems, medium-tech generator systems, and high-tech generator systems. The microwave magnetron operating at a frequency of 2.45 GHz is an example of a low-tech, low-cost, commercially available microwave source that can be operated to generate HPM. The IEMI produced by a low-tech device, an example of which is shown in Fig. 3, was able to cause significant upset or damage to common household appliances [3].

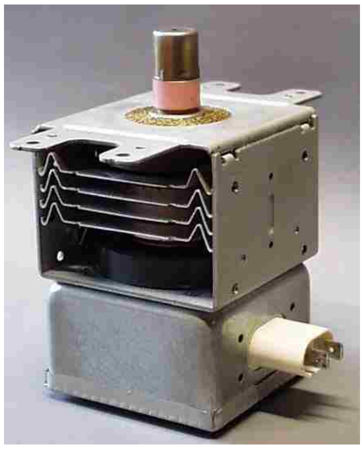


Figure 3: Photograph of a standard household microwave magnetron [4].

Radio Research Instruments provides several medium-tech HPM sources that can be modified to create IEMI generators. An illustration of a medium-tech HPM sources is shown in Fig. 4.

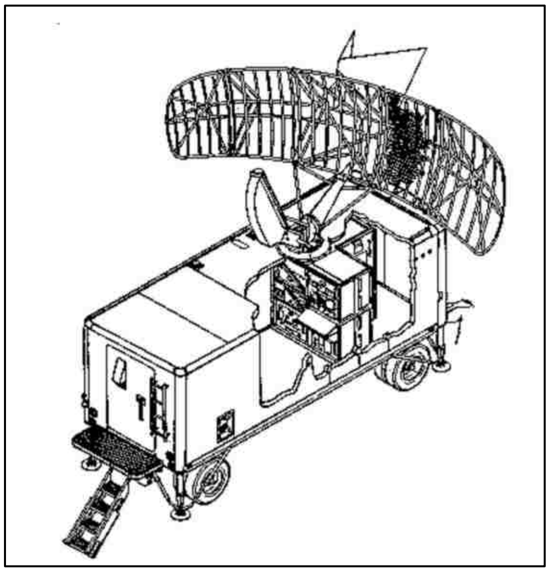


Figure 4: Example of a medium-tech HPM source [5].

Medium-tech generator systems require the skills of a qualified operator and are built using more sophisticated components, such as commercially available radars that can be modified to create IEMI. In contrast, high-tech HPEM generator systems require advanced and specialized technologies and may have the capability to be tuned to cause damage to various targets. An example of such a system is shown in Fig. 5.

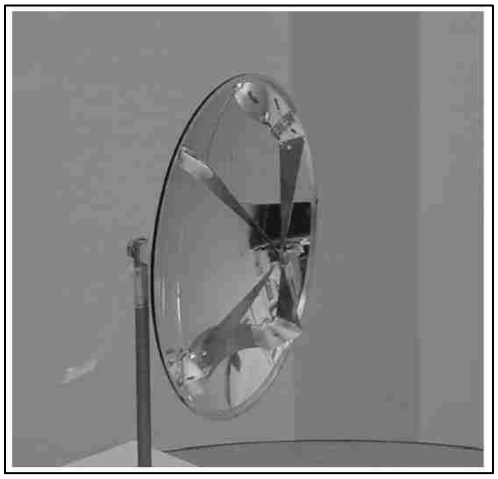


Figure 5: Example of a high-tech HPM system: an impulse radiating antenna (IRA) hyperband source [3].

Finally, classifying IEMI environments by what effects they have on electronic systems is the most appropriate criterion related to this research. Once the EM energy penetrates the electronic system, the effect can manifest itself in several different ways: as noise, as false information, as transient upset, or as permanent damage.

Noise usually couples via front door mechanisms, meaning through receivers that are designed to accept and transmit EM signals. Receivers in civilian electronics systems are designed to operate with electric field intensities as low as several $\mu\mathrm{V/m}$ with a narrow bandwidth. The operation of such receivers is easily degraded through significant presence of noise, but the degradation of operation usually lasts only as long as the noise signal persists.

False information effects occur with electric field strengths of a decade or more above the signal level [3]. Consequences of this upset are more significant, because the electronic systems and the user may not be aware that there is a malfunction in progress, potentially leading to catastrophic failures. False information upsets are usually caused by EM energy that is coupled via front door mechanisms.

Transient upsets usually require several volts of induced signals to affect the logic state of digital systems. The characteristics of the aggressor signal depend on the operating frequency and the quality factor Q of the system, among other factors. Consequences of this type of upset depend on the system design and recovery mechanisms in place to correct transient failure events. This type of upset generally manifests itself through back-door coupling of EM energy into electronic systems. This type of upset and its predictive modeling is the primary focus of this research [3].

The most severe type of effect is permanent damage, an example of which is shown in Fig. 6. Permanent damage results from back door coupling of high-amplitude EM energy into electronic systems. For this effect to occur, there must be a significant overvoltage applied to semiconductor junctions. Typical values of electric field intensity required to cause this type of damage lie in the range of 15-20 $\mathrm{kV/m}$ [6].

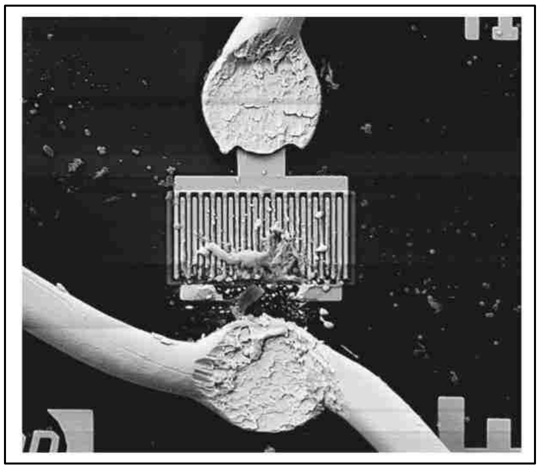


Figure 6: An example of permanent damage to an input transistor due to IEMI [6].

To fully understand how electronic devices operate in extreme EM environments and to improve their immunity, it is important to develop an understanding of what IEMI/EMI is and how it affects electronic systems. IEMI effect on digital electronics is a high priority research topic for the Department of Defense in general, and the Air Force Research Laboratory in particular.

The ultimate goal of this research is to develop knowledge about the effects of HPM on digital systems and to develop a model that can accurately characterize and predict the circumstances under which these events are likely to occur. However, developing a system-level model for IEMI effects on a large-scale system, such as a personal computer, is a very complex task. The number of transistors, the large number of propagation paths through a complex system, in addition to the large number of possible interpretations of transistor states at the software level, make the task of developing a system-level model to predict system malfunction extremely challenging [7].

When it comes to research of IEMI effects on electronic systems, Schamiloglu et al. define an initial framework composed of categories and consequences of electronic effects. Such a framework, presented in Table 3, provides an excellent starting point to guide initial experiments with the ultimate goal of developing a predictive model for upset in electronic systems due to IEMI.

Table 3: Categories and consequences of electronic effects [1].

Failure Mode	**Power Required**	**Wave Shape**	**Recovery Process**	**Recovery Time**
Interference / disturbance	Low	Repetitive pulse or continuous	Self recovery	Seconds
Digital upset	Medium	Short pulse, single or repetitive	Operator intervention	Minutes
Damage	High	UWB or narrowband	Maintenance	Days

To simplify system analysis and provide a good starting point for the development of a predictive model for upset due to IEMI, this research attempts to construct a computational model for prediction of a microcontroller unit (MCU) upset using machine learning tools. The MCU problem presents an ideal choice because it is a complete computer on a single chip that represents an intermediate level of system complexity between an individual CMOS device and a complete digital system [7]. Machine learning tools are well-suited to analyze MCU upset data because of the probabilistic nature of the problem and the necessity to predict such probabilistic upset events. The availability and structure of such data make it an ideally suitable target for machine learning tools.

To date, there have been very limited reports of modeling IEMI effects on electronic systems. This research proposes a novel application of machine learning tools to study transient effects on digital electronics, starting with the most compact representation of a complete digital system – an MCU. Specifically, three machine learning techniques and their applicability to the MCU upset problem are utilized: support vector machines (SVMs), artificial neural networks (ANNs), and Gaussian processes (GPs) for machine learning (GPML).

Several reports have been published covering the application of SVMs to EM problems. For SVM classifiers the key underlying notion is the kernel trick. The kernel trick is used to lift the data into feature space, in which a maximum margin hyperplane is constructed that separates the data into two classes. The popularity of SVMs is due to its appealing features and its effectiveness. Specifically [8]:

a. SVMs represent a methodology where geometric intuition, elegant mathematics, theoretical proofs, and practical algorithms exist;

b. SVMs are a methodology that can be applied to many types of regression, classification, and novelty detection problems;
c. The underlying theory of SVMs is very general and flexible;
d. In contrast to NNs, SVMs eliminate the problem of local minima when trying to minimize an error function;
e. This method is relatively simple to use, and it does not require extensive expert knowledge of the underlying theory;
f. There are many successful applications of SVMs where they have proven to be relatively insensitive to noise as well as to many different distributions of the data used during the training phase.

Feedforward ANNs, such as multilayer perceptrons (MLPs) and SVMs, are popular tools for nonlinear regression and classification problems. From a Bayesian perspective, the choice of an ANN model may be viewed as defining a prior probability distribution over nonlinear functions, and the ANN learning process can be interpreted in terms of the posterior probability distribution over the unknown function. Concisely, Neal [9] has shown that certain ANNs with one hidden layer converge to a GP prior probability distribution over functions, meaning that GPs can be considered as alternatives to ANNs.

A feedforward neural network, also known as an MLP, is a series of logistic regression models stacked on top of each other, with the final layer representing another logistic regression model or a linear regression model, depending on whether the problem to be solved is of the classification or regression type. ANNs have been the subject of great interest due to their suitability to build learning machines. The field is widely viewed as having begun with McCulloch and Pitts [10], who devised a simple mathematical model of the neuron in 1943, in which they approximated the output as a weighted sum of inputs that are passed through a threshold function. Rosenblatt [11] invented the perceptron learning algorithm in 1957, paving the way to estimate the parameters of a McCulloch-Pitts neuron. In 1986, Rumelhart, Hinton, and Williams discovered the backpropagation algorithm, which allows the fitting of models with hidden layers [12].

The GP approach defines a prior probability distribution over functions that can be converted to a posterior over functions once the algorithm is exposed to some training data. A GP assumes that the input probability distribution is jointly Gaussian, with a given mean and covariance function represented by a covariance matrix, or kernel, the requirement of which is to be positive semi-definite. The basic idea is that if the kernel deems two input entries to be similar, then the output of the function at those points will be similar as well. GPs can be thought of as a Bayesian alternative to the kernel methods, such as the SVM method. Although SVMs are sparser and therefore faster, they do not produce a well-calibrated probabilistic output, which is important for many applications.

In summary, communication systems, sensors, data processing devices, and other civilian and military systems make up critical modern infrastructure in daily life or on the battlefield. Damage or failure of these systems have the potential to damage economies and cause injuries or loss of life. Additionally, technological advances and easy access to HPM source and radiating technologies by potentially nefarious actors necessitate research into the effects of IEMI on electronic systems and its components. The assessment of electronic upset and damage are important research activities in the development of behavioral as well as predictive models that will aid in the development of appropriate protection methods for civilian and military electronic systems. Three machine learning methods will be utilized to accomplish the difficult task of modeling MCU upset within this framework, SVM, ANNs, and GPs.

The primary and original contribution of this research is the application of the three machine learning methods (SVMs, ANNs, GPs) to model and predict MCU upset, which is defined as any operation of an MCU other than the one desired. Even though SVMs, ANNs, and GPs have been extensively used to tackle traditional problems in electromagnetics, their applicability to model IEMI effects on electronic devices has not been studied to date. Once the proof-of-concept is achieved, the ultimate goal and related future work will be to generalize these techniques to model upset of more complex electronic systems.

This dissertation is organized in the following manner:

Chapter 1 - Introduction and Research Contributions (Significance of Research): This chapter gives a definition and overview of the intentional EM interference and environment (IEMI/IEME) and why it is important to study this topic. Additionally, it presents key research contributions and their significance.

Chapter 2 - Literature Review: This chapter surveys the state-of-the-art literature in IEMI modeling and machine learning applications to problems of EM nature, including IEMI.

Chapter 3 - Theoretical Considerations: This chapter presents an overview of the theoretical foundations of machine learning, specifically, SVMs, ANNs, and GPs.

Chapter 4 - Experiment Setup and Description: This chapter illustrates the experimental setup, description of key hardware components, as well as a brief description of experiment control and analysis software.

Chapter 5 - Results and Analysis: This chapter presents the results and analysis of the experimental data and a discussion related to any discrepancies observed.

Chapter 6 - Summary and Conclusion: A summary and conclusion of the research are presented in this chapter.

Chapter 7 - Future Research: This chapter makes recommendations for future research related to the topic of IEMI modeling with machine learning tools.

Appendices: The appendices contain data that is pertinent to this research, but that was omitted from the main body of the document to avoid clutter.

Chapter 2: Literature Review

The purpose and organization of this chapter are twofold. The first part surveys the literature on IEMI research, while the second part covers the literature on the use of machine learning techniques, specifically ANNs, SVMs, and GPs applied to problems related to IEMI modeling and other problems related to electromagnetics, as applicable.

There has been significant research into effects of EM energy on the operation of electronic systems, individual components, as well as microcontrollers. For instance, Wendsche et al. [13], [14] have found that the immunity of digital equipment in general, and MCUs in particular, largely depends on the hardware and software structure. They investigated the time-dependent immunity of 8-bit and 16-bit microcontrollers by subjecting the MCU power supply to single burst spikes of EM energy and have found that susceptibility to IEMI depends directly on the microcontroller's operational functions, especially on the microinstructions used for an assembler instruction. In other words, they have shown that the MCU susceptibility is directly related to the instruction that the MCU is executing and the instruction-dependent hardware state at the time of impulsive disturbance. Additionally, they have found that the susceptibility (or probability of upset) of one assembler instruction can vary anywhere from 0% to 100% depending on when the EM energy is injected into the device. They found that their results did not depend on the particular model or type of microcontroller used, but that it was applicable universally across all the devices they tested, which is why this research is focused on studying one microcontroller model.

Additionally, Wang et al. [15] used the direct injection method to inject an RF signal into the clock pin of an 8-bit ripple counter that was designed and fabricated using

AMI 0.5 pm process technology. In this study, the authors have shown that relatively low levels (16.8 dBm) of injected pulse-modulated RF signal into the clock network affected the normal functioning of a digital counter circuit. The reason for the effect manifestation at a relatively low RF power level is the decreasing feature size that results in a lower charge that is required for transistors to change states. At a carrier frequency of 1 GHz, the end-to-end system was not perfectly matched, as they determined the magnitude of the reflection coefficient to be approximately $\Gamma \approx 0.559$, so not all power was delivered to the clock network. They intend to further investigate additional parameters, such as RF pulse width, pulse rise/fall time, and varying input voltage bias to see how they affect and cause state changes at output pins.

In their paper, Nitsch et al. [16] give an overview of the susceptibility of a wide range of electronic systems such as computer networks, computer systems, microprocessor boards, microcontrollers and other integrated circuits (ICs) to IEMI. They also assessed physical damage to the devices under test (DUT) by opening them and examining the damaged components. The experiment consisted of free-field tests in which the equipment under test (EUT), enclosed in a TEM structure, was subjected to an EM free field of varying strength. It should be noted that the EUT was fairly dated (e.g. 233 MHz motherboards, 10/100 Ethernet connections, among others).

The portion of their experiment that is most pertinent to this research is their investigation of microcontroller upset. The parameters varied during their research are shown in Fig. 7, and a summary of results is shown in Table 4. Their measures of upset are: breakdown failure rate (BFR), defined as the number of breakdowns of the system divided by the number of pulses applied to it; the breakdown threshold (BT), defined as the value of the electric field strength at which the BFR attains 5% of maximum; and the breakdown bandwidth (BB), defined as the span of electric field strength at which the BFR changes from 5% to 95% of maximum. The authors tested the effects of different data, reset, oscillator, and power supply line lengths, as well as clock rate on the BFR and BT. They found that different devices of the same MCU type exhibited similar BFR and BT, but different types of MCUs showed different BFRs and BTs. In general, BT of the tested MCUs was heavily dependent on the reset line length, clock, and power supply line length, but it was independent of data line length and clock rate (up to 8 MHz).

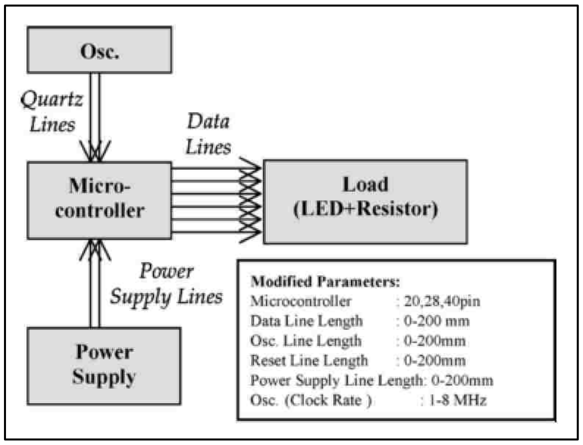
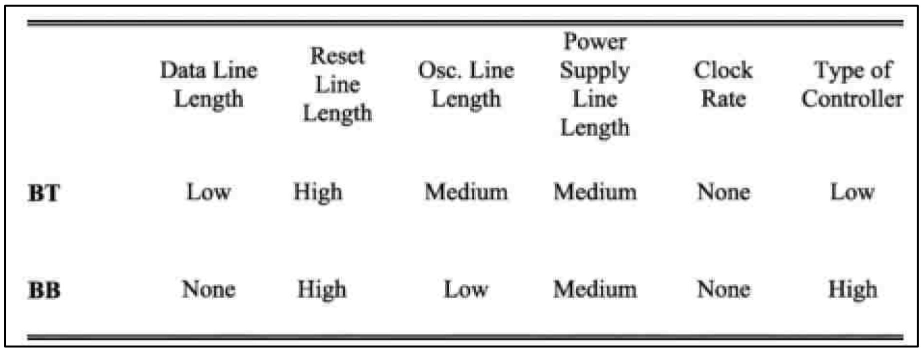

Osc.
Quartz Lines
Data Lines
Micro-controller
Load (LED+Resistor)
Power Supply Lines
Power Supply
Modified Parameters:
Microcontroller : 20,28,40pin
Data Line Length : 0-200 mm
Osc. Line Length : 0-200mm
Reset Line Length : 0-200mm
Power Supply Line Length: 0-200mm
Osc. (Clock Rate) : 1-8 MHz

Figure 7: Microcontroller susceptibility test setup [16].

When it comes to BB, it is influenced by the type of microcontroller, reset line length, and power supply line length. BB was not dependent on the oscillator line length, data line length, nor the MCU clock operating frequency. The level of influence of each parameter on BT and BB is summarized in Table 4.

Table 4: MCU susceptibility results summary [16].



	Data Line Length	Reset Line Length	Osc. Line Length	Power Supply Line Length	Clock Rate	Type of Controller
BT	Low	High	Medium	Medium	None	Low
BB	None	High	Low	Medium	None	High

Camp et al. [17] studied the IEMI susceptibility of three different microcontroller types manufactured using complementary metal oxide semiconductor (CMOS) technology, following which they used a novel statistical test procedure to model the breakdown probabilities as a function of interfering pulse amplitude. Their setup is shown in Fig. 8.

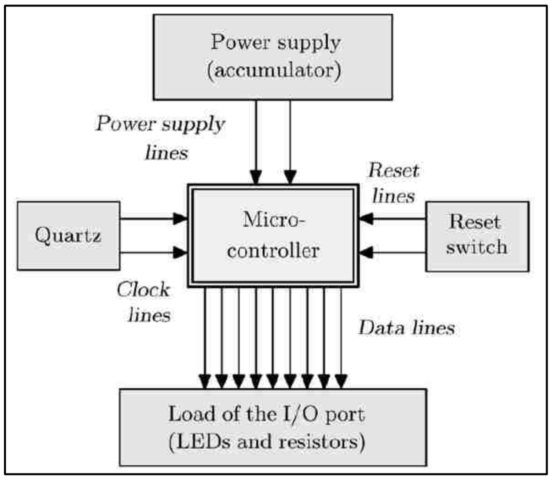
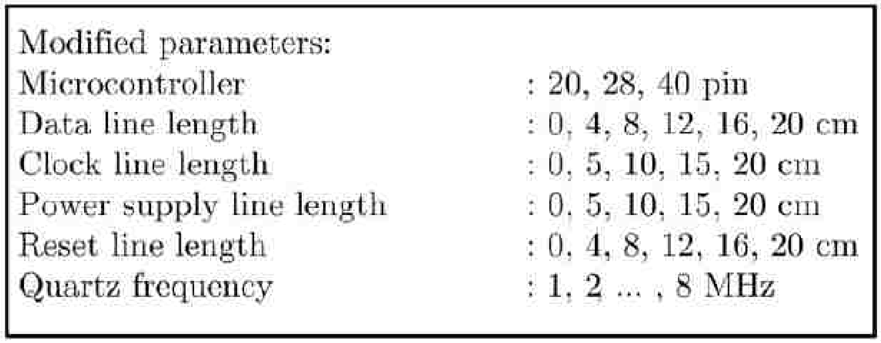


Figure 8: Camp, et al. setup to model MCU breakdown behavior [17].

It should be noted that Camp et al. conducted a free-field test while modifying the experimental parameters shown in Fig. 9.



Modified parameters:	
Microcontroller	: 20, 28, 40 pin
Data line length	: 0, 4, 8, 12, 16, 20 cm
Clock line length	: 0, 5, 10, 15, 20 cm
Power supply line length	: 0, 5, 10, 15, 20 cm
Reset line length	: 0, 4, 8, 12, 16, 20 cm
Quartz frequency	: 1, 2 ... , 8 MHz

Figure 9: Test parameters for modeling of MCU breakdown behavior [17].

Similar to [16], Camp et al. studied the probability of effect (PoE) as a function of the I/O port state, different signal line lengths, MCU clock rate, and EM pulse shape. The results of their study are summarized in Fig. 10.

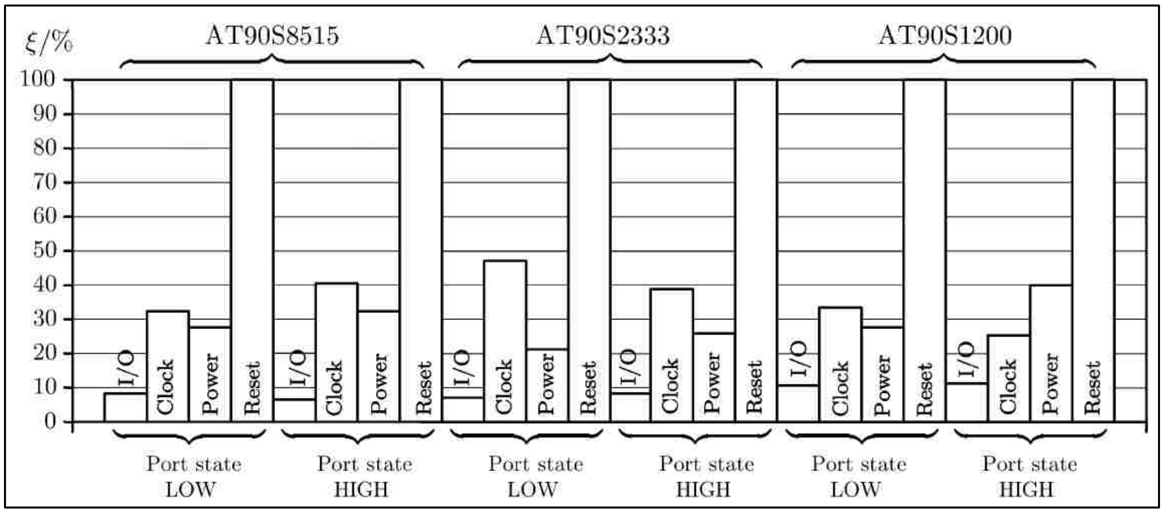


Figure 10: Camp et al.'s results of IEMI breakdown study [17].

The symbol ξ is a descriptive factor that accounts for the length of the signal line. Their results show that the PoE is highest when the IEMI affects the reset pin, followed by the clock and power pins, depending on the port state. IEMI effect on the I/O pin remains relatively constant. Further analysis of their data showed that for a Weibulldistributed critical EM field strength, it is possible to determine the breakdown failure probability of the MCU under test.

Investigating the behavior of electronic devices when they are subjected to IEMI is a relevant and vibrant research topic of interest to various civilian and government agencies all over the world. Understanding how devices operate in aggressive EM environments helps in the development of effective countermeasures and hardening methods. In addition to understanding how RF transients affect electronic devices at the device physics level, it is also important to build knowledge on how device level effects propagate to the functional layer, i.e. at the software level. Yuan [18] presents a process for building a model of how conducted EMI (cEMI) causes software-level effects. In their approach, a black box cEMI modeling procedure is utilized. This model building approach relies on only on measurement information. The model relies on the premise that the internal impedance of the MCU changes as different instructions are being executed and that each instruction corresponds to a unique internal impedance (intZ) signature as well as an internal current activity (intCA) signature. The result is that the BBIR algorithm can estimate intZ and intCA for a specific instruction with a reasonable degree of accuracy, as shown in Fig. 11.

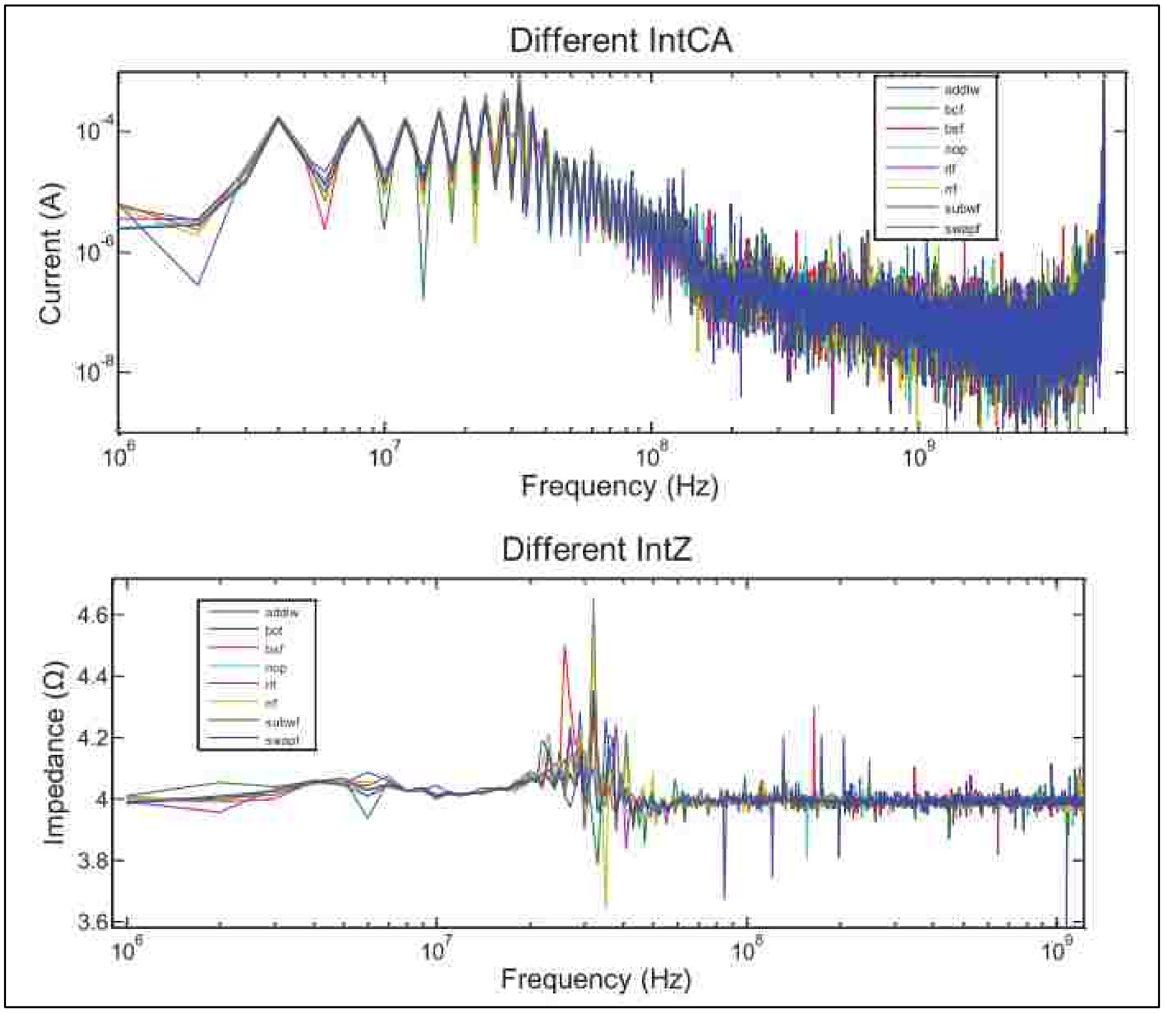


Figure 11: Internal impedance and internal current activity for different MCU instructions [18].

In conclusion, the author states that the BBIR model can be used to accurately estimate the cEMI of code sequences or even entire programs being executed by the MCU.

Complex problems in science in general, and electromagnetics in particular, increasingly require a multidisciplinary approach to solve. Traditionally, the electromagnetics and machine learning disciplines have been considered to be two distinct scientific fields, even though researchers in the field of electromagnetics have leveraged the diverse toolset of the machine learning community. This chapter presents an overview of successful applications of machine learning techniques, such as NNs, SVMs, and GPs to problems in the electromagnetics domain.

In their paper, Caorsi and Cevini [19] utilized an ANN approach to solve an EM diagnostic problem and predict the EM field absorption inside a dielectric. They utilized the values of the incident electric field at a fixed number of physical locations as the input training vector and features describing the EM absorption as the output of the ANN. Research of this type has the potential to predict the absorption of EM fields inside biological tissues that are exposed to wireless communication systems. The estimation in this research is performed by a single-layer feedforward perceptron ANN with sigmoid activation trained using a backpropagation algorithm. Their preliminary results show that this approach provides a satisfactory accuracy of the ANN in reconstructing the selected absorption features and that the ANN can successfully estimate these characteristics. Good results have been obtained for signal to noise ratios (SNR) of 30 dB and above.

Additionally, Micu et al. [20] successfully utilized a layer recurrent ANN solution for an EMI problem, wherein they studied induced AC voltages in the underground metallic pipeline due to nearby high-voltage grids, as shown in Fig. 12. This approach used 37 training instances that were developed using traditional computational electromagnetics methods, as shown in Table 5. Their research concluded that the ANN-based approach proved to be very effective in predicting the induced voltages. The ANN approach is advantageous because it provides nearly instantaneous results when compared to traditional finite element methods (FEM) [20]. Good results were obtained, despite using a relatively small number of training events.

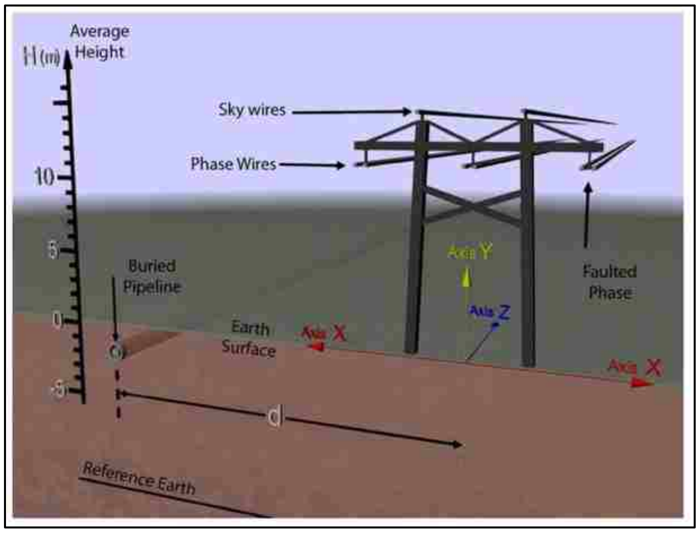


Figure 12: Candidate scenario for an NN application to an electromagnetics problem [20].

Table 5: Neural network training database [20].

No	*d* [m]	*x* [m]	*y* [m]	ρ [Ωm]	MVP Amp. 10^{-5} [Wb/m]	MVP Phase [°]
1	70	70	-15	30	36.1	-22.8
2	100	100	-30	30	29.9	-31.23
3	800	770	-30	30	4.23	-82.64
4	800	785	0	30	4.27	-78.83
5	1000	1030	-15	30	2.48	-90.27
6	2000	1970	-22.5	30	0.476	-108.1
7	2000	2020.69	-8.61	30	0.436	-108.54
8	400	384.81	-7.82	70	17.2	-44.46
9	400	424.77	-6.93	70	15.8	-46.72
10	1000	970	-15	70	5.95	-73.04
11	1000	1007.5	0	70	5.68	-72.98

Ceperic and Baric [21] present a simple and efficient method of modeling the immunity of ICs to conducted EMI using ANNs. In their work, they showed that the universal approximation property of ANNs can efficiently determine the conducted EM immunity behavior. In their approach, they use an input sinusoid with a specified voltage that is applied to the input pin of the IC as a training vector. The measured output voltage at each output pin is used as a target output vector to train the NN that was developed using the MATLAB Neural Network Toolbox. Their setup is shown in Fig. 13.

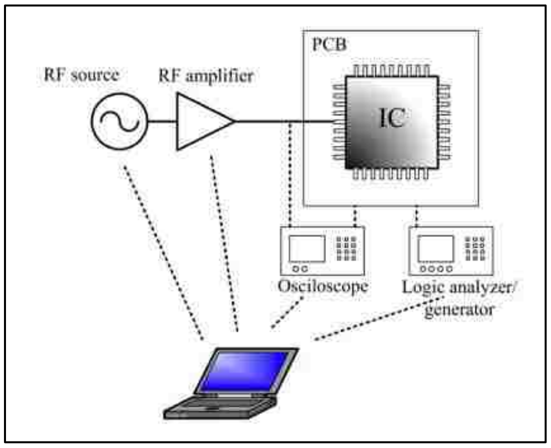


Figure 13: Test setup to determine electromagnetic immunity behavior [21].

In contrast to Micu et al. who utilized only 37 training instances, Ceperic and Baric noted that a large number of input-output learning pairs needed to be obtained to train the ANN ($n > 10{,}000$) successfully. The reason for this large discrepancy could lie in the fact that Ceperic and Baric only used one dimension (input voltage) to train the ANN, effectively utilizing a 1xN training vector, whereas Micu et al. used a 5xN training vector. Micu et al. also utilized two different layer-recurrent NNs, which they claim eliminated their need for a large training data set. Another interesting contrast between the two approaches used to gather training data is that Micu et al. used a (FEM)-based approach to generate training data, whereas Ceperic and Baric chose an experimental approach, both with good results and execution times much faster than when compared to traditional computational methods.

In their paper, Li et al. [22] utilized ANNs to solve electromagnetic compatibility (EMC) problems in electronic systems design. Specifically, an ANN was used in a classic EMI-type problem to predict the crosstalk coupling between two wires. Fig. 14 shows a schematic of their setup.

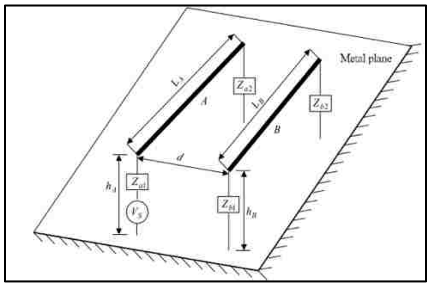
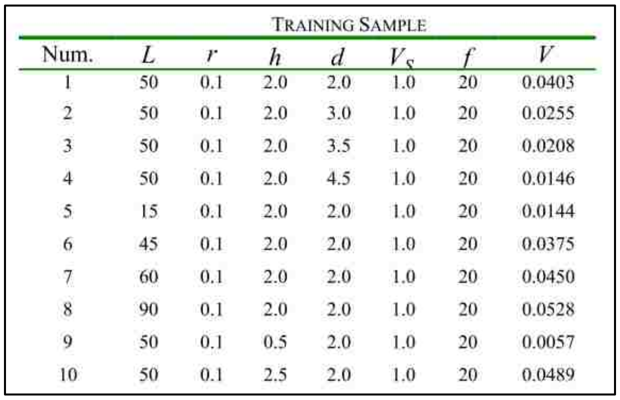


Figure 14: Multiconductor transmission line (MTL) crosstalk model [22].

According to EM field theory, the coupled voltage on the victim line is a function of several parameters, including wire geometry, height above the metal plate, the separation between lines, and aggressor voltage. Table 5 shows the values of the parameters that have been used to train and test the NN. Their results obtained via ANNs was in good agreement with results produced by multiconductor transmission line (MTL) theory.

Table 6: MTL scenario training vector [22].



TRAINING SAMPLE

Num.	L	r	h	d	V_S	f	V
1	50	0.1	2.0	2.0	1.0	20	0.0403
2	50	0.1	2.0	3.0	1.0	20	0.0255
3	50	0.1	2.0	3.5	1.0	20	0.0208
4	50	0.1	2.0	4.5	1.0	20	0.0146
5	15	0.1	2.0	2.0	1.0	20	0.0144
6	45	0.1	2.0	2.0	1.0	20	0.0375
7	60	0.1	2.0	2.0	1.0	20	0.0450
8	90	0.1	2.0	2.0	1.0	20	0.0528
9	50	0.1	0.5	2.0	1.0	20	0.0057
10	50	0.1	2.5	2.0	1.0	20	0.0489

Chahine et al. [23] have successfully used NNs to model the susceptibility of ICs to conducted EM disturbances. Traditional modeling of IC susceptibility to continuous wave (CW) interference, as proposed in EMI/EMC literature, is accomplished via two well-established methods. The first method consists of re-use of the IC emission model (ICEM) model for predicting IC susceptibility by substituting the internal structure of the

IC with a load based on lumped elements. The second method makes use of input/output buffer information specification (IBIS) files and Spice technological data to prepare a preliminary susceptibility model. Each of these two traditional approaches requires internal knowledge of the IC under test which may be kept confidential by the manufacturers. This paper takes a three-step approach to model building using NNs. In step 1, they apply the direct power injection method according to IEC 62132-4, which is an EMC testing standard. In step 2, they observe the IC response to the injected signal and record data. In step 3, they implement the ANN approach to extract the mathematical expression that relates the input vector and the output vector. They achieved near-perfect prediction capability.

Next, Devabhaktuni et al. [24] demonstrate an ANN-based approach for rapid EMI/EMC analysis of printed circuit boards (PCB) and shielding enclosures. They hypothesize that ANNs can be viable alternatives to physics-based simulation, driven by ANNs' generalization and extrapolation capabilities. They term their proposed approach as "reverse-modeling." It entails interchanging input and output columns during NN training activities, and they report errors of <5% in their attempt to model results obtained using MTL theory. The error is computed with respect to results that obtained using standard EM field solvers such as HFSS.

As illustrated in several examples above, NNs have been successfully utilized in solving a variety of problems related to electromagnetics. In these examples, small sets of diverse data were used to train the NN, which produced results that rivaled traditional computational methods used in electromagnetics in a fraction of the computational time.

In an overview of SVM applications to electromagnetics, Oliveri et al. [8] present some cases of SVM solutions to EM-related problems along with a discussion on the potential and limitations of SVM applications in electromagnetics. Specifically, they demonstrate the utility of SVM regression to model the direction of arrival (DoA) of signals impinging on antenna arrays with very good results. Furthermore, they utilize SVM classification to detect buried objects, and although the method did not correctly determine the number of buried objects, it was able to estimate with good accuracy the position of the objects in the soil.

In addition to the NN-based approach to EMI/IEMI modeling, Ceperic and Baric [25] published results where they transformed a traditional cEMI problem into a binary classification problem that is solved by SVMs. In their approach, the black-box SVM model is used to predict the conducted immunity when the circuit is exposed to cEMI interferences and the SVM results are compared with transistor-level simulations obtained with SPICE. The setup of their approach is modeled according to the IEC 62132-4 standard and is shown in Fig. 15.

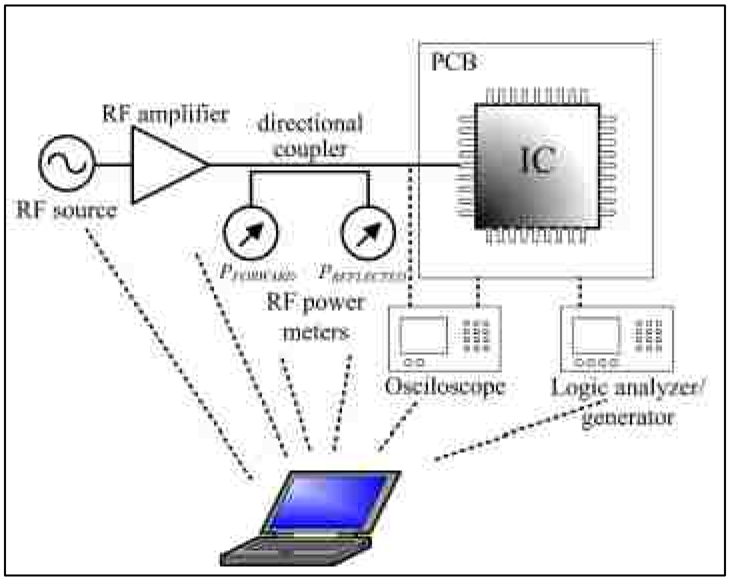


Figure 15: Direct power injection method according to IEC 62132-4 [26].

The number of training data used for this approach was 10,000, and it should be noted that the SVM testing condition differed significantly from the SVM training condition, indicating that the SVM generalized very well to unseen data. Indeed, Ceperic and Baric were able to obtain testing accuracy of almost 100%, in addition to a much-improved speed-up factor of 10 over traditional SPICE models, showing the SVM feasibility to model cEMI of ICs to be feasible with excellent accuracy.

While Ceperic and Baric used SVM to solve a cEMI problem that was formulated in a binary classification manner, Wu et al. were able to solve a traditional EM problem using SVMs for regression. Specifically, they utilized SVM to estimate the permittivity and permeability of materials. In their approach, HFSS was used to obtain the S-parameters of the structure under consideration, from which the permittivity and permeability parameters were derived. The material under test was modeled as a microstrip transmission line, as shown in Fig. 16. It should be noted that they do not mention the number of data points that were used for training or testing, but the accuracy of the SVM to simulate the permeability and permittivity was over 93%. The SVM simulated results vs. HFSS results are shown in Fig. 17.

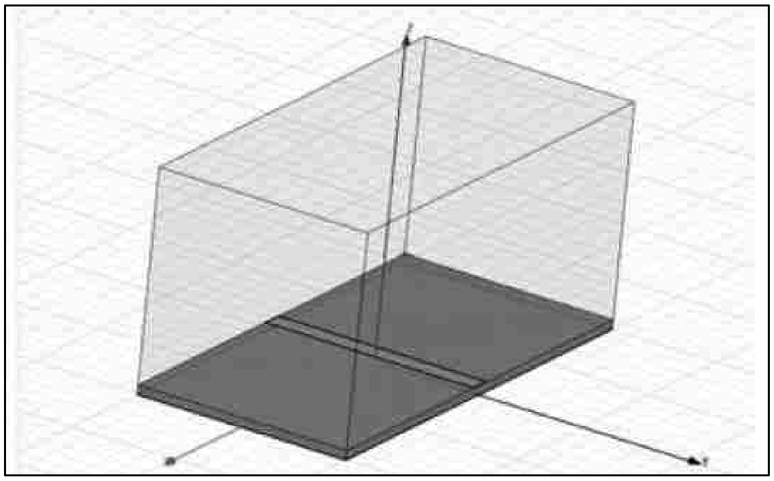


Figure 16: Simulation scenario for SVM-derived electromagnetic properties of the material under test [27].

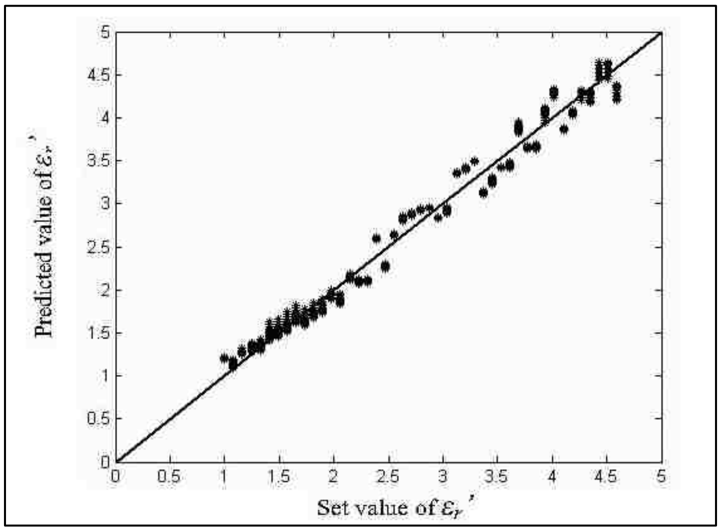


Figure 17: SVM results vs. HFSS results [28].

SVMs' and ANNs' utility to problems in electromagnetics has been successfully demonstrated. However, the utilization of GPs in electromagnetics is not as prevalent. To the author’s knowledge, this is the first time a GP-approach has been taken to study problems in electromagnetics, specifically problems pertaining to EMI.

Chapter 3: Machine Learning Overview

This chapter is composed of four significant subsections. The first subsection presents an overview of the machine learning field and how it is used to solve complex problems for which no analytical solutions exist. The second subsection reviews the theory behind SVMs, while the third and fourth subsections present material related to ANNs and GPs, respectively.

Chapter 3-1: Machine Learning Theory

Tools that aid in the analysis of large datasets are becoming increasingly important in industry. The term “big data” is usually used to describe a scenario or scenarios in which there is a lot of information and data available, but due to its volume, it is very difficult to extract knowledge. A tool that is often used to make sense of a lot of seemingly disconnected data is machine learning. Specifically, machine learning is defined as “a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty” [12], such as which data to collect next!

Machine learning or learning from data is used to solve problems for which no analytic solutions exist, but there is data available to construct an empirical solution based on that data. The idea of machine learning covers a lot of territory and material. It is one of the most widely used techniques applied to problems in science, engineering, and economics, among other fields [29].

Many professionals and researchers in the financial, medical, and social media fields all have successfully utilized machine learning techniques to solve very complex and important problems. The power of learning from data is that the entire process of determining the most likely function that explains the data and makes a prediction on future data combinations is fully automated. In other words, the learning algorithm reverse-engineers important factors that make up the function that is based solely on previous data.

Machine learning uses a specific set of terms to formalize the learning problem. The input to the machine learning algorithm is labeled as $\boldsymbol{x}$, while the unknown target function $f\colon X \to Y$ uses the set of all possible inputs $\boldsymbol{x}$ (X) to produce the desired output space Y. That is the crux of the learning problem. Additionally, there is a data set D that represents all the possible input-output examples, i.e. $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$, where $y_N = f(\boldsymbol{x}_N)$ for $N = 1, \ldots, N$. These examples are also referred to as data points or data that has been collected from the experiment with the corresponding outcome. Finally, there is a learning algorithm that uses the data set D to determine a formula g such that $g\colon X \to y$ thus approximating f. The algorithm chooses the approximating function g from a set of candidate functions under consideration, labeled as a hypothesis set H. For example, the algorithm can consider a set of all linear models H and pick the best linear model fit to the data. A workflow representation of the formalized learning problem is shown in Fig. 18. The result of the machine learning algorithm will be only as good as the extent to which g faithfully replicates f when it is presented with new data, or data that the algorithm has not seen before.

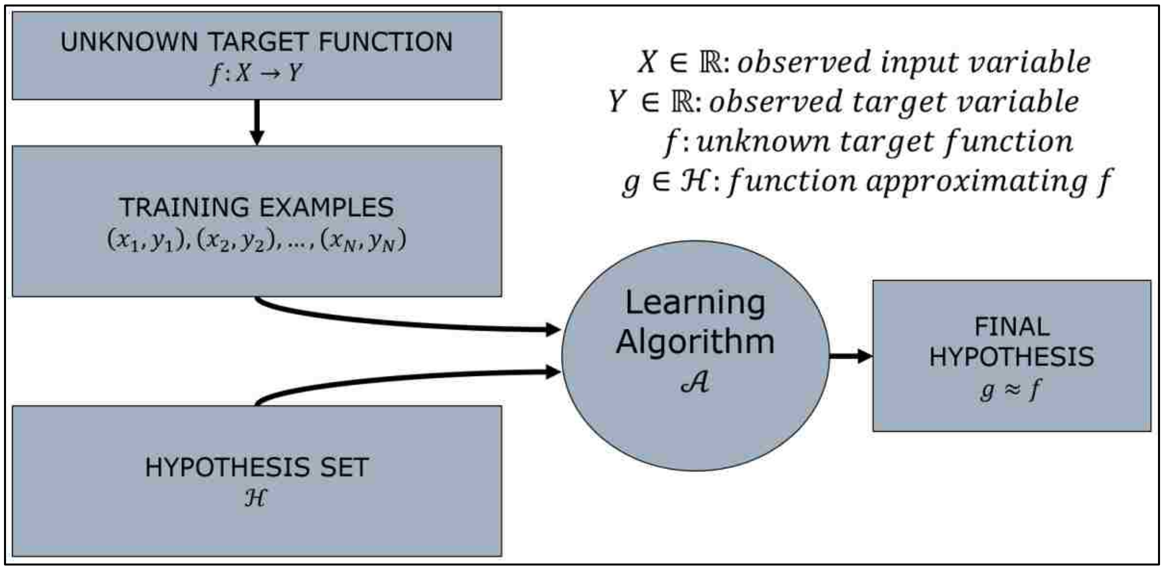

UNKNOWN TARGET FUNCTION
$f: X \to Y$
TRAINING EXAMPLES
$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
HYPOTHESIS SET
$\mathcal{H}$
Learning Algorithm
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
$X \in \mathbb{R}$: observed input variable
$Y \in \mathbb{R}$: observed target variable
f: unknown target function
$g \in \mathcal{H}$: function approximating f

Figure 18: Basic flow of the learning problem [29].

To achieve a good approximation for f, the learning algorithm chooses g that best matches f based on the previously available data with the goal that it will still continue to match f for new data; in other words, the goal is that g will generalize well [29].

The field of machine learning involves many probability models and algorithms that are suitable for a wide variety of data and tasks. Even though the presence of numerous models and algorithms may make the machine learning field convoluted, the field can be divided into two main types: supervised and unsupervised learning.

In the predictive or supervised learning approach, the learning goal is to learn a mapping, or a function, from inputs x to outputs y, given a labeled training set of input-output pairs represented by $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ where N represents the number of training samples. In the simplest machine learning scenario, each training input x_i is a $D-$ dimensional vector of real numbers, representing measurements or attributes that relate to the experiment, such as the dimensions or descriptions of some object, or as in this case RF-pulse attributes, RF pulse injection locations, and so on. These are also called features, attributes, or covariates describing some phenomenon or experiment measurements [12].

Similarly, the form of the output or response variables can in principle be anything, but most machine learning methods require y_i to be a categorical or nominal variable from some finite set, $y_i \in (1, \dots, C)$, i.e. a label representing a binary outcome of a random experiment, i.e. "upset" or "no upset". In the case that y_i is categorical, the machine learning problem is usually referred to as a problem of classification or pattern recognition and when y_i is a real-valued scalar such as a voltage level, current level, or magnitude of a magnetic field, the problem is considered to be of a regression type. In the unsupervised learning approach, we are given inputs such as $\mathcal{D} = \{(x_i)\}_{i=1}^{N}$ with the goal to find interesting patterns in the data. This approach is also referred to as knowledge discovery. Problems of this type do not have predetermined outcomes and generally do not have obvious error functions for the algorithm to minimize, making these types of problems less well defined [12]. There are other learning types available, such as reinforcement learning, but that type of learning is not the focus of this research and will not be covered in detail.

Since we have experiment outcomes that can be used as labels, i.e. "upset"/"no upset," the focus of this research will be on supervised learning. Within supervised learning techniques there exist two types of problems to be solved: classification and regression. In classification, we wish to learn a mapping from inputs x to outputs y, where $y \in \{1, \dots, C\}$, with C being the number of classes according to which experiment outcomes are classified. The case where $C = 2$ is called binary classification. In binary classification, it is often assumed that $y \in \{0,1\}$ or $y \in \{-1,1\}$, depending on the implementation of the machine learning algorithm. The case where $C > 2$ is called multiclass classification or multi-label classification depending on whether the class labels are mutually exclusive or not [12].

Definition (Learning problem) The learning problem is to determine the functional relationship $h \in \mathcal{Y}^{\mathcal{X}}$ between objects $x \in \mathcal{X}$ and targets $y \in \mathcal{X}$ based solely on a set of example scenarios $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) = \big((x_1, y_1), \dots, (x_m, y_m)\big) \in (X \times Y)^m$ of size $m \in \mathbb{N}$ which are drawn from an unknown distribution $\boldsymbol{P}_{XY}$ and are independent and identically distributed. If the output space contains a finite number $|Y|$ of elements then the task is called a classification problem.

Chapter 3-1-1: Probabilistic Perspective on Machine Learning

Probability theory is sometimes the most suitable approach for some data analysis problems. Probability theory can be applied to any problem involving uncertainty. In the field of machine learning, uncertainty is present in the form of several questions: what is the best model to explain a set of data? What is the best prediction about a future event given some past data? What measurement should be performed next? The application of probability theory to problems involving machine learning is closely related to the field of statistics, with slightly different focus and terminology [12]. More formally, it is possible to infer something outside of the data set D using only D, but in a probabilistic way [29].

The following example illustrates the role of probability theory in machine learning, and it is covered in more detail in [29]. The example illustrates whether choosing a sample from a set and whether we can say something about the objects outside of that sample. Consider a bin containing many red and green balls. The proportion of red and green balls in the bin is such that if we pick a ball at random, the probability that it will be red is μ and the probability that it will be green is $1 - \mu$. It is assumed that we do not know the value for μ. Additionally, suppose a random sample of N independent balls is picked with replacement from this bin, and the fraction v of red balls is observed. Is it possible to infer something about the value of μ given v? This situation is similar to taking a poll and the probability distribution of the random variable v in terms of the parameter μ is well understood. When μ the sample size is large, v tends to be close to μ.

The relationship between v and μ can be quantified by a bound called the *Hoeffding Inequality*. It states that for any sample size N,

$$\mathbb{P}[|v - \mu\ | > \varepsilon] \leq 2e^{-2\varepsilon^2 N} \text{ for any } \varepsilon > 0, \tag{1}$$

where $\mathbb{P}[\cdot]$ denotes the probability of an event with respect to the random sample we pick and where ε is any positive real value. Informally, as the sample size N grows, it becomes exponentially unlikely that v will deviate from μ by more than a chosen tolerance ε. The only quantity that is random is v which depends on the random sample or experiment outcome. The utility of the Hoeffding Inequality is to infer the value of μ using the value of v, although it is μ that affects v and not vice versa. Although $\mathbb{P}[|v - \mu\ | > \varepsilon]$ depends on μ, the probability can be bound by $2e^{-2\varepsilon^2 N}$, which does not depend on μ. In order to make a good approximation of μ using v, a larger sample size N is needed to make $2e^{-2\varepsilon^2 N}$ sufficiently small. We do not have control over v since v is based on a particular hypothesis $\mathcal{H}$. During the learning process, we explore an entire hypothesis set $\mathcal{H}$, looking for some $h \in \mathcal{H}$ that has a small error rate. If only one hypothesis is available, then we are not really learning but simply verifying whether that hypothesis is good or not [29].

In general, when dealing with data that is generated by a random experiment or ambiguous cases in which it is difficult to determine a classifier with absolute certainty, it is necessary to handle predictions in terms of a probability value. A probability distribution over possible classifiers or labels given the input vector x and a training set D can be defined as $p(y|\boldsymbol{x}, D)$. Generally, this represents a vector of length C – in the MCU upset scenario $C = 2$ and in this case, it is sufficient to return the single number $p(y = 1|\boldsymbol{x}, D)$, since $p(y = 1|\boldsymbol{x}, D) + p(y = 0 \text{ or } - 1|\boldsymbol{x}, D) = 1$, where $y = 1$ and $y = 0\ or\ - 1$ denotes MCU upset and no MCU upset respectively. This notation states that the PoE is conditional on the test input $\boldsymbol{x}$ and the training set D, and implicitly the model M being considered. The setup from Fig. 18 has been updated to reflect the role of probability, and the new setup is shown in Fig. 19.

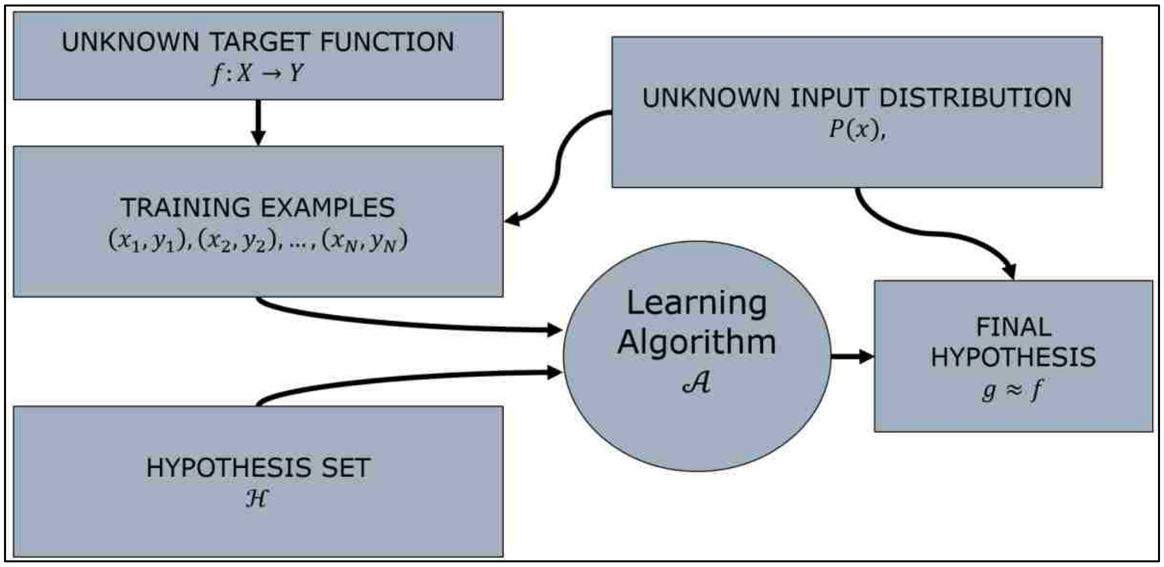


Figure 19: Basic learning problem flow with probability factored in [29].

Concisely, the question of whether D tells us anything outside of D that we did not know before has two possible but different answers. If the requirement is to produce an accurate, deterministic answer then the answer is no, D will not tell us anything outside of D that we did not know before. On the other hand, if a probabilistic answer will satisfy the requirement, meaning that the data set D tells us something that is likely about f outside of D, then the answer is yes, the data set D will tell us something outside of D that was not known before [29].

Chapter 3-1-2: Error and Noise in Data

There are two very important concepts related to the machine learning problem that resemble the real world. The first concept is the meaning of approximation when it is claimed that the chosen hypothesis approximates the target function well. The second concept is related to noise that makes the output of f not uniquely determined by the input data. For example, identical input data can produce both outputs in a binary classification problem. In this particular case, identical experiment parameters can produce “upset” at one time and “no upset” at other times. This section examines ramifications of having a noisy target in connection with the learning problem.

Machine learning techniques are not expected to replicate the target function perfectly. The chosen hypothesis g is only an approximation of f; hence, it is expected that there is some error involved. This error measure needs to be sufficiently quantified so that it enables a quantitative assessment of how far the hypothesis g is from the actual target function f. The choice of the error measure is an important step as it directly affects the outcome of the learning process. A different choice of error measure may lead to different final hypotheses even if the target and the data are the same, hence the importance of error measure selection [29].

Formally, an error measure in the context of machine learning quantifies how well each hypothesis h in the model approximates the target function f, i.e.

$$\text{Error} = E(h, f). \tag{2}$$

While $E(h, f)$ is entirely based on h and f, it is almost universally defined based on the errors on individual input points in $\boldsymbol{x}$. In some respects, the error measure will depend on the nature of the problem and the data it contains. For example, if a pointwise error is defined, the overall error is going to be the average value of the pointwise error. So far, an error of the following form has been described: $E(h(\boldsymbol{x}), f(\boldsymbol{x})) = [\![h(\boldsymbol{x}) \neq f(\boldsymbol{x})]\!]$. Ideally, the error measure should be user specified, as it heavily depends on the nature of the learning problem. Another way to express the notion of the error measure is the cost of using h when f should be used instead [29]. Generally, the choice of error measure depends on the how the system is going to be used and not on any inherent criterion that can be independently determined during the learning process.

When it comes to the effect of noise in many practical applications of machine learning, the data that is used for learning is usually not produced by a deterministic target function. Instead, the data are generated in a noisy environment in which the output is not uniquely determined by the input. In the scenario of MCU upset, two identical sets of criteria can produce different system behavior, i.e. “upset” or “no upset.” Therefore, the function that determines the upset model will not be a deterministic function, but a probabilistic and noisy one, hence the need for the probabilistic treatment of machine learning as discussed earlier.

Noisy data can be readily modeled within the machine learning framework. Instead of considering $y = f(\boldsymbol{x})$, a random variable y can be considered. This random variable will be affected rather than determined by the input vector $\boldsymbol{x}$**.** In other words, we would have a target distribution $P(y|\boldsymbol{x})$ instead of a target function $y = f(\boldsymbol{x})$. In a sense, a noisy target can be considered as a deterministic target with added noise. If y is a real-valued target, the expected value of y given $\boldsymbol{x}$ can be considered the deterministic $f(\boldsymbol{x})$ with noise added in the form of $y - f(\boldsymbol{x})$. There is no loss in generality if we consider the target function f to be a distribution rather than a function. Hence, the entire analysis of the feasibility of learning applies to noisy target functions as well. This is because the Hoeffding Inequality applies to an arbitrary, unknown target function. The general conclusion is that noisy targets are not as easy to learn as deterministic ones [29]. The workflow of the general learning problem, incorporating an error measure, is shown in Fig. 20.

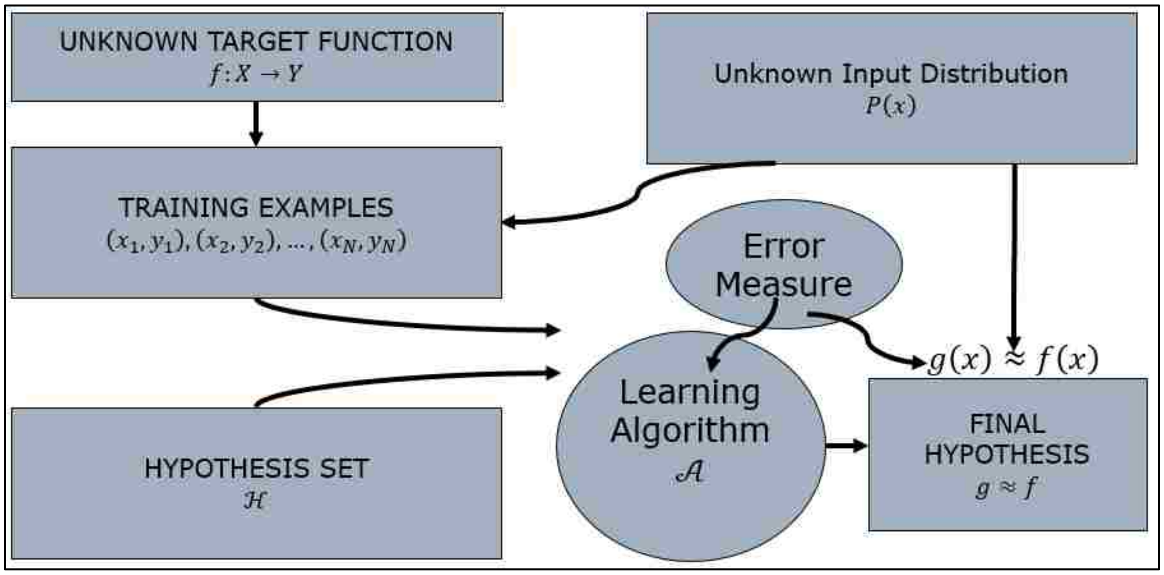


Figure 20: Basic learning problem flow with error measure incorporated [29].

As mentioned earlier, the choice of the error measure depends on how the machine learning system is going to be used rather on some inherent criterion determined through the learning process, and in this case, a pointwise error comparing each prediction (upset/no upset) with the particular measurement of upset/no upset.

Chapter 3-1-3: Algorithm Generalization Theory

The ability of the learning model to perform well on data that the algorithm has not seen before is referred to as generalization. Before covering the theory behind generalization, it is important first to introduce some definitions. First, the error rate within the sample, which corresponds to v in the previously considered bin example, is called the *in-sample error*, or fraction of D where f and h disagree,

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N} [\![h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)]\!]. \tag{3}$$

Similarly, the out-of-sample error is defined as

$$E_{\text{out}}(h) = \mathbb{P}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})], \tag{4}$$

which corresponds to μ in the bin model. This probability is based on the distribution P over X used to sample the data points $\boldsymbol{x}$**.** In other words, the out-of-sample error E_{out} is used to measure how well the training on the data set D has generalized to data that the algorithm has not seen before, while E_{in} is based on data points that have been used to train the algorithm [29].

Since the concept of generalization is key in machine learning, the *generalization error* can be defined as the discrepancy between E_{in} and E_{out} and the Hoeffding Inequality provides a way to characterize it with a probabilistic bound, such that

$$\mathbb{P}[\![|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon]\!] \leq 2Me^{-2\epsilon^2 N} \tag{5}$$

for any $\varepsilon > 0$. In other words, if we pick a tolerance level δ, i.e. $\delta = 0.05$ and assert with probability at least $1 - \delta$ that the out-of-sample error

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)} \tag{6}$$

representing the generalization bound that bounds E_{out} by E_{in}. The Hoeffding Inequality introduced earlier applies to this generalization bound [29].

The error bound $\sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)}$ depends on M, which is the size of the hypothesis set $\mathcal{H}$, which can be a set of infinite size, meaning that the bound goes to infinity and becomes meaningless. It turns out that a lot of interesting learning models have an infinite set of hypotheses, including the ANN model and GP model. It is desirable to replace M with a finite set so that the error bound becomes meaningful. The way M is obtained in the first place was by taking the disjunction of events, such as

$$|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \varepsilon \qquad \text{or} \tag{7}$$

$$|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \varepsilon \qquad \text{or} \\ \vdots \tag{8}$$

$$|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \varepsilon, \tag{9}$$

which is guaranteed to include our chosen hypothesis g that approximates f, i.e. $|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon$ since g is always one of the hypotheses in $\mathcal{H}$ [29].

It turns out that the events $|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \varepsilon,$ where $m = 1, \dots, M$ are strongly overlapping if h_1 is similar to h_2, which is usually the case. In fact, in a typical learning model, many hypotheses are indeed very similar. Consider the perceptron model as an example, as the weight vector $\mathbf{w}$ is varied during training, it is possible to generate infinitely many hypotheses that differ from each other by an infinitesimal amount. The mathematical theory of generalization is dependent on this observation.

If machine learning techniques are to be successfully used in practical applications, it is necessary to determine the effective number of hypotheses to be examined by the learning algorithm. The determination of the number of hypotheses to be examined is done by specifying a growth function, which will replace M in the generalization bound above. The growth function is a combinatorial quantity that captures how different the hypotheses in $\mathcal{H}$ are. To show how the growth function helps us determine an upper bound on the number of hypotheses to be tested, it is necessary to define the growth function, bound its value, and use it to replace M in the generalization bound above.

Let $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in X$. The dichotomies generated by $\mathcal{H}$ on these points are defined by $\mathcal{H}(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N) = \{(h(\boldsymbol{x}_1), \dots, h(\boldsymbol{x}_N)) | h \in \mathcal{H}\}$. The growth function is defined for a hypothesis set $\mathcal{H}$ by $m_{\mathcal{H}}(N) = \underset{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in X}{max} |\mathcal{H}(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)|$, where $|\cdot|$ denotes the cardinality (number of elements) of a set. In other words, $m_{\mathcal{H}}(N)$ is the maximum number of dichotomies that can be generated by the hypothesis set $\mathcal{H}$ on any N points. Just like M, $m_{\mathcal{H}}(N)$ is a measure of the number of the number of hypotheses in $\mathcal{H}$, except that a hypothesis is not considered on N points instead of the entire set X, and it is bounded by

$$m_{\mathcal{H}}(N) \leq 2^N \text{ [29].} \tag{10}$$

Chapter 3-1-4: Is Learning Feasible?

Having defined the learning model and associated error functions, the next most logical question to ask is: is learning feasible for the problem at hand? In typical machine learning problems, the target function f is the objective of the learning algorithm with the most important assertion that f is truly unknown. In that case, how can a limited data set reveal enough information to determine the entire target function? There is a possibility that more than one function f can explain the underlying data, however that same function would sometimes indicate an outcome synonymous with “upset” and at other times an outcome synonymous for “no upset”, for the same input data! In other words, how is it possible to infer something outside of the data set D using only D with certainty? It turns out that this scenario is the rule, and not the exception in the field of machine learning [29].

There appear to exist conflicting arguments about the feasibility of learning. One argument says that it is not possible to learn anything outside of D and the other says that it is possible. If a probabilistic view of learning is adopted, then we get a positive answer to the question of feasibility without paying a high price. The only assumption that is required for this to work is that the data in D are generated independently. There is no need to use any particular probability distribution or even the need to know what distribution is used. However, the probability distribution needs to remain constant from the generation of the data samples in D through the evaluation of how well g approximates f. Keeping the probability distribution consistent throughout the learning process is what makes the Hoeffding Inequality applicable [29].

The second concept related to the feasibility of learning is the error measure. The process of learning produces a hypothesis g that is supposed to approximate the unknown target function f reasonably well, in the case that learning is feasible. If the application of machine learning algorithms is successful, then $E_{\text{out}}(g) \approx 0$. This is not the case for probabilistic analysis. In probability-based learning, we have $E_{\text{in}}(g) \approx E_{\text{out}}(g)$, meaning that in order for $E_{\text{out}}(g) \approx 0$, it is necessary that $E_{\text{in}}(g) \approx 0$. There is no guarantee that we will find a hypothesis that achieves $E_{\text{in}}(g) \approx 0$, however we will know when we find it. In essence, the feasibility of learning boils down to two key questions: first, is it possible to ensure that $E_{\text{out}}(g) \approx 0$ is close enough to $E_{\text{in}}(g) \approx 0$? Second, is it possible to make $E_{\text{in}}(g)$ small enough. The Hoeffding Inequality addresses the first question. The answer to the second question is obtained after the learning algorithm processes the data and the value of $E_{\text{in}}(g)$ is determined. Analyzing the feasibility of learning in this manner provides important insight into the role that different components of the learning problem play [29].

Two of the more important components of machine learning are the hypothesis set $\mathcal{H}$ and the target function f. If the number of hypotheses M increases, the risk of $E_{\text{in}}(g)$ being a poor estimator of $E_{\text{out}}(g)$ does as well. In this case, M represents the complexity of $\mathcal{H}$, and it needs to be kept as simple as possible, even though the most interesting solutions to machine learning problems present themselves when $\mathcal{H}$ is complex to an extent. Additionally, a complex $\mathcal{H}$ gives more flexibility in finding an approximation g that fits the data well, so a tradeoff between complexity and good approximations is necessary. It seems obvious that a complex target function f is more difficult to learn than a simple one. Additionally, if the target function is complex, the second question of feasibility of learning comes into play, because $E_{\text{in}}(g)$ is likely to be larger than desired. The good news is that most target functions in reality are not very complex, so it is possible to learn them from a reasonable set of training data D using a reasonable set of hypotheses $\mathcal{H}$ [29].

Chapter 3-1-5: Important Learning Principles

The field of machine learning incorporates three general and very important principles. The first principle relates to the choice of model and is commonly referred to as Occam's razor. The second principle is related to the data and is known as sampling bias. The third principle is also related to data handling and is called data snooping. A good understanding of these three principles increases the likelihood that the machine learning approach is implemented correctly and the generalization of performance is interpreted properly.

Occam's Razor dates back to the 14th century and is attributed to William of Occam. The term razor is meant to trim down the explanation to the bare minimum that is consistent with the collected data. The penalty of model complexity is, in a way, a manifestation of Occam's razor. In other words, Occam's razor reinforces the principle that "the simplest model that fits the data is also the most plausible" [29]. Applying this principle means that we should choose as simple of a model as we think we can get away with to explain the data.

Two natural considerations are relevant in applying this principle. First, we need to establish what it means for a model to be simple and second, we need to convince ourselves (and perhaps others) that a simpler model yields a better solution. One of the more common ways to describe object complexity is in terms of the number of bits required to describe an object in the computational world. In terms of simple counting, a simple object is not only intrinsically simple, but it is also one of few because there are fewer objects that have short descriptions than there are objects that have long descriptions. The second consideration that simpler is better has nothing to do with elegance or aesthetics; it means that a simpler model has a better chance of being correct for a given data set. Stated differently, a large set of complex hypotheses would determine a fit for a dataset D, even if the relationship between input and target vectors in D is completely random. Fitting the data in such a way does not mean much in the context of machine learning [29].

The second important principle of machine learning is related to sampling bias. It states that “if the data is sampled in a biased way, learning will produce a similarly biased outcome” [29]. Applying this principle to the machine learning problem means that the probability distribution used for training must be the same probability distribution used for testing or performance evaluation of the machine learning algorithm and model.

The final principle that applies to all problems involving machine learning is commonly referred to as data snooping. Simply, this means that the machine learning algorithm must not be exposed to the same data that is used for training and validation. Otherwise, the outcome is compromised. That is a bad thing. One way to avoid this pitfall is to choose the learning model before seeing any of the data. Some aspects to consider in making the choice of learning model can be the number of data points and prior knowledge regarding the input space and target function, just not the actual data set D [29]. The most common way to fall for this type of pitfall is to reuse the same data set during training, validation, and testing.

Chapter 3-2: Support Vector Machines (SVMs)

In machine learning problems, our goal is to discover structure in the data. This section describes how to discover structure in the data using SVMs. Since the early 1990s, there have been a lot of theoretical developments and applications of SVMs, the theory of which is based on the combination of optimization theory, statistical learning, kernel theory, and algorithm development. SVMs are a popular tool used to solve problems in fields such as text categorization, biomedical engineering, magnetic resonance imaging, linear signal processing, speech recognition, and image processing, but it is not as prevalent in the field of electromagnetics as its promising potential might suggest. Christodoulou and Martinez-Ramon outline four typical situations in which SVMs would make good candidates for use in the field of engineering electromagnetics. The use of SVMs for analysis is suitable under the following circumstances: (1) when closed forms solutions to the problem do not exist; (2) when practical applications require real-time or near real-time performance; (3) when faster convergence rates and smaller errors are required in the optimization of large systems; and (4) sufficient measurement data exist to train an SVM for prediction purposes [30].

Support Vector Machines (SVM) are machine learning systems that utilize a hypothesis space of linear functions in a high-dimensional feature space. SVMs are trained with a learning algorithm used in optimization theory. This learning algorithm implements a learning bias derived from statistical learning theory. The learning strategy behind this algorithm was introduced by Vapnik in the early 1970s and even though it has only gained traction recently, it is a very powerful method that has already outperformed most other learning systems in a wide range of applications. This theoretical summary in this section closely follows Christianini and Shawn-Taylor, whose text can be referenced for more detail regarding SVMs and other kernel methods [31].

In supervised learning, the machine is given a training set of examples (inputs) with associated labels (targets/output values). The examples from which the machine learns are in the form of attribute vectors that makes the input space a subset of $\mathbb{R}^n$. With the availability of attribute vectors, several sets of hypotheses can be chosen to model the problem at hand. Among all the available hypotheses, linear functions represent the simplest possible classifier.

Binary classification is usually performed by using a real-valued function $f: X \subseteq \mathbb{R}^n \to \mathbb{R}$ as follows: the input sample $\boldsymbol{x} = (x_1, \dots, x_n)^T$ is assigned to a positive class $(+1)$ if $f(\boldsymbol{x}) \geq 0$, and otherwise to the negative (-1) class. In the case where $f(\boldsymbol{x})$ is a linear function of $\boldsymbol{x} \in X$, it can be written as

$$f(\boldsymbol{x}) = \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = \sum_{i=1}^{n} w_i x_i + b, \tag{11}$$

where $(\boldsymbol{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ represent the parameters that control $f(\boldsymbol{x})$ that must be learned from the data. A geometrical representation of this type of hypothesis is that the input space X is separated into two parts by the hyperplane $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = 0$, as shown in Fig. 21.

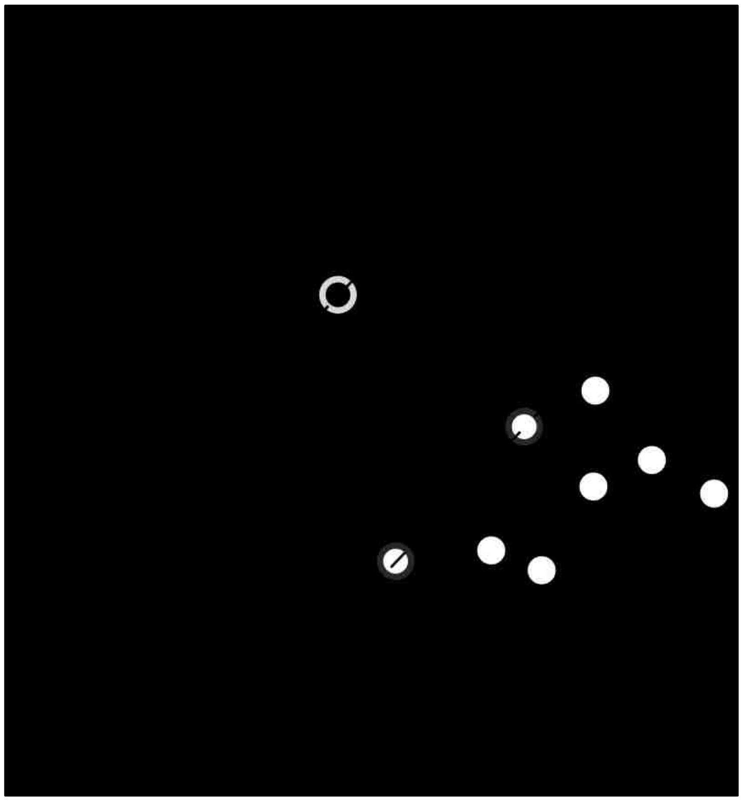


Figure 21: A separating hyperplane $(\boldsymbol{w}, b)$ for a two-dimensional training set [32].

A hyperplane is defined as an affine subspace of dimension $n - 1$ that divides a space into two parts. These two spaces depend exclusively on the input vector of input data. As seen in Fig. 21, the vector **w** defines a direction perpendicular to the hyperplane and the value of b moves the hyperplane parallel to itself. In the SVM approach, it is necessary to include $n + 1$ free parameters if it is desired to represent all possible hyperplanes in $\mathbb{R}^n$. SVMs can be utilized when the experiment data has exactly two classes, as is the case in the MCU upset problem (0: no upset; 1: upset). The SVM classifies data by finding the most optimal hyperplane that separates all data points of one class from those of the other class. The best and most optimal hyperplane for an SVM means the hyperplane with the largest margin between the two classes. The margin means the largest width of the slab that is parallel to the hyperplane and that has no interior data points. Like in the neural network approach, the quantities **w** and b are referred to as weight and bias and, the linear classifier in SVMs can be referred to as a linear discriminant and is nearly equivalent to the perceptron concept in NNs.

The problem of learning a hyperplane that separates two sets of points is an ill-posed problem, because several different solutions may exist. Some algorithms, such as the perceptron algorithm, may be susceptible to producing different results based on the order in which the data are processed. In short, the danger of arriving at solutions for ill-posed problems is that those solutions may not be equally useful. Awareness of this potential pitfall and ways of circumventing it helps to produce better learning solutions. One way to rectify an ill-posed problem such as this one is to try to optimize a different cost function, ensuring that the solution the algorithm arrives at is always unique. In this case, a solution can be chosen that not only separates the two classes of data but also to choose a hypothesis that is located at a maximum distance from all data points. This solution is a hyperplane that realizes the maximal margin as defined by $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = 1$ and $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = -1$, as seen in Fig. 21. This solution also possesses the characteristic of maximum stability [31].

The SVM binary classification problem has two important cases. The first case deals with perfectly separable data in which a most optimal hyperplane can separate the data into two classes with no error. This approach has a primal and a dual formulation from a mathematical perspective. The second case deals with data that is not perfectly separable into two classes, i.e. there are some data points that will be misclassified. Both cases can be mathematically expressed in primal and dual form.

For the perfectly separable case, the data for training is a set of vectors $\boldsymbol{x}_j$ along with their categories or targets $\boldsymbol{y}_j$. For some dimension d, $\boldsymbol{x}_j \in R^d$, and $\boldsymbol{y}_j = \pm 1$. In the mathematical primal formulation, the equation for the separating hyperplane becomes

$$f(x) = x'\beta + b = 0, \tag{12}$$

where $\beta \in R^d$ and b is a real number. The problem that is formulated in terms of the best separating hyperplane goes as follows: find β and b that minimize $\|\beta\|$ such that for all data points $(\boldsymbol{x}_j, \boldsymbol{y}_j)$

$$\boldsymbol{y}_j f(\boldsymbol{x}_j) \geq 1. \tag{13}$$

In this case, the support vectors are the $\boldsymbol{x}_j$ on the boundary that separates the two classes, i.e. those for which $\boldsymbol{y}_j f(\boldsymbol{x}_j) = 1$.

For the case where the experiment data is not perfectly separable (non-separable), the SVM algorithm can use a soft-margin, meaning that the hyperplane will separate many, but not all data points. For SVMs, there are two standard formulations of soft margins, and both involve the slack variable $\boldsymbol{\xi}_j$ and a misclassification penalty parameter, or so-called box-constraint C.

The L^1 – norm problem is formulated as

$$\min_{\beta,b,\xi}\left(\tfrac{1}{2}\boldsymbol{\beta}'\boldsymbol{\beta}+\boldsymbol{C}\textstyle\sum_j\boldsymbol{\xi}_j\right) \text{ such that } \boldsymbol{y}_j f\left(\boldsymbol{x}_j\right)\geq 1-\boldsymbol{\xi}_j;\ \boldsymbol{\xi}_j\geq 0, \tag{14}$$

and the L^2 – norm is formulated as

$$\min_{\beta,b,\xi}\left(\frac{1}{2}\boldsymbol{\beta}'\boldsymbol{\beta}+\boldsymbol{C}\sum_j\boldsymbol{\xi}_j^2\right) \text{ such that } \boldsymbol{y}_j f\left(\boldsymbol{x}_j\right)\geq 1-\boldsymbol{\xi}_j;\ \boldsymbol{\xi}_j\geq 0. \tag{15}$$

Furthermore, dual representation is a very important concept in linear machines, and it is present in a wide class of algorithms, including SVMs. A very important property of the dual representation is that the data only appear through entries in the Gram matrix and not through their attributes. Similarly, in the dual representation of the decision function $f(\boldsymbol{x})$, it is only the inner products of the data with the new test point that are needed. Many of the algorithms and concepts relevant to SVMs are special cases of optimization problems, for which a mathematical framework exists that encompasses the concept of duality [31].

In mathematical dual formulation, which represents the dual quadratic programming problem, which is computationally less complex, the L^1-norm problem becomes

$$L_p = \frac{1}{2}\boldsymbol{\beta}'\boldsymbol{\beta} - \sum_j \boldsymbol{\alpha}_j \left(\boldsymbol{y}_j(\boldsymbol{x}_j'\boldsymbol{\beta} + \boldsymbol{b}) - 1\right), \tag{16}$$

where $\boldsymbol{\alpha}_j$ are the positive Lagrange multipliers. The objective is to find a stationary point L_p over $\boldsymbol{\beta}$ and $\boldsymbol{b},$ so setting the gradient of L_p to 0 we obtain

$$\boldsymbol{\beta} = \sum_j \boldsymbol{\alpha}_j \boldsymbol{y}_j \boldsymbol{x}_j \tag{17}$$

and

$$0 = \sum_j \alpha_j y_j. \tag{18}$$

Substituting these values into L_p, the dual equivalent L_D is obtained:

$$L_D = \sum_j \alpha_j - \frac{1}{2}\sum_j \sum_k \alpha_j \alpha_k y_j y_k \, x_j' x_k, \tag{19}$$

which is maximized over $\alpha_j \geq 0$. In practice, however, many Lagrange multipliers have a zero maximum value. The nonzero α_j in the solution to the dual problem define the hyperplane, which gives $\beta = \sum_j \alpha_j y_j x_j$, and the data points x_j that correspond to the nonzero values of α_j are the support vectors. Additionally, the derivative of L_D with respect to a nonzero Lagrange multiplier is zero at an optimum, resulting in [33]

$$y_j f(x_j) - 1 = 0. \tag{20}$$

In the case of nonseparable data, the soft-margin approach is used. Here the goal is to minimize the following function for the L^1 – norm problem:

$$L_p = \frac{1}{2}\beta'\beta + C\sum_j \xi_j - \sum_j \alpha_j \left(y_j f(x_j) - (1 - \xi_j)\right) - \sum_j \mu_j \xi_j. \tag{21}$$

Complex, real-world applications like the MCU upset modeling problem, usually require more extensive hypothesis spaces than simple linear functions offer, thus requiring the learning to be performed in kernel-induced feature spaces, in which the algorithm can exploit abstract features of the data. Another way to look at kernel-induced feature spaces is to express the data as a linear combination of the attributes, however more complex feature spaces are required for the learning algorithm to be effective in practical applications. Kernel representations of the data offer an alternative solution by projecting the data into a high-dimensional feature space helping to increase the computational power of linear learning machines. The use of learning machines in dual representation is what makes the implicit utilization of this step possible. As mentioned earlier, in dual representation the data examples never appear isolated, but occur rather as inner products between pairs of examples. The advantage of the dual representation is that the number of tunable parameters does not depend on the number of attributes present in the data. By replacing the inner product with an appropriate kernel function, the nonlinear mapping of data to a high-dimensional feature space is accomplished implicitly, thereby increasing the number of tunable parameters, assuming that the kernel computes the inner product of the input feature vectors corresponding to the input pair.

When learning in the feature space, the complexity of the target function to be learned depends on the way it is represented. This representation should ideally match the specific learning problem, so a common strategy in machine learning is to change the representation of the input data, i.e.

$$\boldsymbol{x} = (x_1, \dots, x_n) \rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = \big(\phi_{\mathbf{1}}(x), \dots, \phi_{\mathbf{1}}(x)\big). \tag{22}$$

This step is equivalent to mapping the input space X into a new space $F = \{\boldsymbol{\phi}(\boldsymbol{x}) | \boldsymbol{x} \in X\}$.

Mapping the data into another space has been known to simplify the learning task greatly, and that has given rise to a number of techniques used to represent the data in a useful way. While discussing kernel-based techniques such as SVMs, it is useful first to define some terminology used in the field. The quantities that are introduced to describe the data are referred to as features, while original quantities are called attributes, while the task of choosing the most suitable representation is referred to as feature selection. The space X is usually referred to as the input space while $F = \{\boldsymbol{\phi}(\boldsymbol{x}) | \boldsymbol{x} \in X\}$ is called the feature space. Fig. 22 illustrates how a feature map can simplify the classification task.

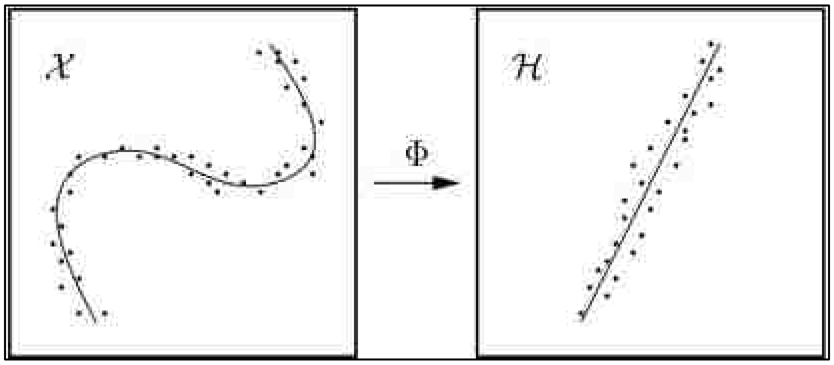


Figure 22: Illustration of feature mapping [34].

There are different approaches to the process of feature selection. The goal of feature selection is to identify the smallest set of features that still conveys the essential information described by the attributes. This process is referred to as dimensionality reduction. More formally,

$$\boldsymbol{x} = (x_1, \dots, x_n) \rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = \left(\phi_{\boldsymbol{1}}(\boldsymbol{x}), \dots, \boldsymbol{\phi}_{\boldsymbol{d}}(\boldsymbol{x})\right), d < n. \tag{23}$$

The process of dimensionality reduction reduces the computational expense and generalization performance of the model because it mitigates effects of the curse of dimensionality that states that a growing number of features have adverse effects on model performance. The curse of dimensionality is a rather unfortunate phenomenon because a diverse set of (potentially redundant) features can potentially improve the probability of finding an $f(\boldsymbol{x})$ with a standardized learning machine [31]. Another component of feature selection is the detection and removal of irrelevant features. One of the main reasons why feature selection is so important in machine learning, is that the use of many features during training can create overfitting problems resulting in poor generalization of the learning model.

Linear SVMs can be applied as a learning method to learn nonlinear relations in data. For this learning to occur successfully, a set of nonlinear features needs to be selected, and the data needs to be represented in a way such that a linear machine can be used. One way to accomplish this is by implicit mapping into feature space. Hence, the set of hypotheses to be considered in this case are of the type

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi_i(\boldsymbol{x}) + b \tag{24}$$

where $\phi: X \rightarrow F$ is a nonlinear map from the input space to a feature space, meaning that the use of linear machines is composed of two steps: first, input data is mapped into a feature space suitable for linear machines and second, a linear machine is used to classify the data in the feature space.

The reason for this two-step process involving linear machines is that currently, only the linear hyperplane optimality criterion is available, enabling the practical application of SVMs. Currently, there are no general results available for nonlinear functions. Fortunately, Mercer's theorem, developed in the early 1900s is key to extending the learning principle of SVMs to nonlinear machines, and it is key to validating a kernel function. Concisely, vectors $\boldsymbol{x}$ in a finite dimension input space can be mapped to a possibly infinite-dimensional Hilbert space through implicit mapping as discussed earlier [30].

The dual representation property of linear learning machines means that the hypothesis can be expressed as a linear combination of the data points used during training. This way, the decision rule can be evaluated using inner products between the training points and the test point:

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i y_i \langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b\,; \tag{25}$$

moreover, if there is a way to compute the inner product directly $\langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle$ in feature space directly as a function of the original input points, then a nonlinear learning machine has effectively been created. This method of direct computation of the inner product of training data in feature space and testing data is called a kernel function. Formally, a kernel function is a function K, such that for all $\boldsymbol{x}, \boldsymbol{z} \in X$

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \rangle, \tag{26}$$

where $\boldsymbol{\phi}$ represents a mapping from X to an inner product feature space F [31].

The name kernel comes from the integral operator theory, which forms the foundation of the theory of the relation between kernels and their corresponding feature spaces. With the use of dual representation, the dimension of the feature space does not affect the computation complexity of the learning problem. By accomplishing implicit mapping into feature space, the linear machine is trained in that same space, sidestepping the computational complexity of having to evaluate the entire feature map. The only representation of the training data that is used in SVMs is their representation in the form of a Gram matrix in feature space. The Gram matrix, with the entries specified by $\boldsymbol{G} = \left(\langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle\right)_{\boldsymbol{i,j=1}}^{\boldsymbol{l}}$, is the only way the data enter the learning algorithm. This Gram matrix is also referred to as the kernel matrix [31].

The symbol $\boldsymbol{K}$ used to denote a kernel function and the key to the kernel-based learning approach is to find a kernel function that can be evaluated efficiently. Once such a function has been determined, the decision rule can be evaluated by at most l evaluations of the kernel

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\boldsymbol{x}_i \boldsymbol{x}) + b. \tag{27}$$

Because kernel functions are critical to the performance of SVMs, it is important to determine what characterizes them and consequently build kernels that may be a better representative of the data to which the learning machine is applied.

The use of a kernel function presents an alternative and attractive computational short-cut. Generally, to use this promising approach, a complicated feature space would need to be created, along with a definition of inner product in that space. The final step would be to compute the inner product in the space using the original inputs. In practical applications, these steps are accomplished by the definition of a kernel function directly, making this a very critical step during the kernel-based learning approach.

Several characteristics are important in determining whether a kernel is valid for some feature space. First, the function must be symmetric, i.e.

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}) \rangle = \langle \phi(\boldsymbol{z}) \cdot \phi(\boldsymbol{x}) \rangle = K(\boldsymbol{z}, \boldsymbol{x}); \tag{28}$$

second, the function must satisfy the following inequalities that follow from the Cauchy-Schwartz inequality:

$$\begin{aligned} &K(\boldsymbol{x}, \boldsymbol{z})^2 = \langle \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}) \rangle^2 \leq \|\phi(\boldsymbol{x})\|^2 \|\phi(\boldsymbol{z})\|^2 \\ &= \langle \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}) \rangle \langle \phi(\boldsymbol{z}) \cdot \phi(\boldsymbol{z}) \rangle \\ &= K(\boldsymbol{x}, \boldsymbol{x}) K(\boldsymbol{z}, \boldsymbol{z}), \end{aligned} \tag{29}$$

none of which is sufficient to guarantee the existence of a feature space [31].

Mercer's theorem, developed in the 1900s, provides a characterization of when a function $K(\boldsymbol{x}, \boldsymbol{z})$ is a kernel. To define a kernel function, consider X to be a finite input space with $K(\boldsymbol{x}, \boldsymbol{z})$ representing a symmetric function on X. Then $K(\boldsymbol{x}, \boldsymbol{z})$ is a kernel function if and only if the matrix

$$\boldsymbol{K} = \left(K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right)_{i,j=1}^{n} \tag{30}$$

is positive semi-definite; in other words, the matrix has non-negative eigenvalues [31].

According to Mercer's theorem, if X is a compact subset of $\mathbb{R}^n$ and suppose that K is a continuous symmetric function such that the integral operator $T_K: L_2(X) \to L_2(X)$,

$$(T_K f)(\cdot) = \int_X K(\cdot, \boldsymbol{x})\, f(\boldsymbol{x}) d\boldsymbol{x} \tag{31}$$

is positive, i.e.

$$\int_{X \times X} K(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{x}) f(\boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{z} \geq 0 \tag{32}$$

for all $f \in L_2(X)$. Then $K(\boldsymbol{x}, \boldsymbol{z})$ can be expanded in a uniformly convergent series on $X \times X$ in terms of $T_K{}'s$ eigenfunctions $\phi_j \in L_2(X)$ with $\left\| \phi_j \right\|_{L_2} = 1$ and with positive associated eigenvalues $\lambda_j \geq 0$,

$$K(\boldsymbol{x}, \boldsymbol{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\boldsymbol{x}) \phi_j(\boldsymbol{z}). \tag{33}$$

The positivity condition $\int_{X \times X} K(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{x}) f(\boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{z} \geq 0$ above corresponds to the positive semi-definite condition in the finite case. Hence, the conditions for Mercer's theorem require that for any finite subset of X, the corresponding matrix is positive semi-definite [31]. The basic idea behind this is that vectors **x** in a finite dimensional input space can be mapped to a possibly infinite-dimensional Hilbert space $\mathcal{H}$ provided with a dot product through linear transformation $\phi(\cdot)$, most of which are unknown. The dot product of the corresponding spaces can be expressed as a function of the input vectors. Specifically, for every Mercer kernel defined over the domain $X \subset \mathbb{R}^d$ there exists a reproducing kernel Hilbert space (RKHS) of functions defined over X for which K is the reproducing kernel [26, 27].

The condition satisfying Mercer's theorem is not easy to prove for any function. Among the first kernels to be proven to satisfy Mercer's conditions were the homogeneous polynomial kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T, \boldsymbol{x}_j)^p \tag{34}$$

and the inhomogeneous polynomial kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T, \boldsymbol{x}_j + 1)^p. \tag{35}$$

The mapping for these two kernels can be determined explicitly, and their corresponding Hilbert space has finite dimension. Another important kernel that applies to this research is the Gaussian kernel (RBF kernel). This kernel is also widely used in many other practical applications and has the form

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{\frac{\|x_i - x_j\|^2}{2\sigma^2}}. \tag{36}$$

For the Gaussian kernel, the nonlinear mapping is not explicit, and the corresponding Hilbert space is infinite-dimensional [30].

The positivity condition $\int_{X\times X} K(\boldsymbol{x},\boldsymbol{z}) f(\boldsymbol{x}) f(\boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{z} \geq 0$ is the key to verifying that a news symmetric function is a kernel, specifically, the requirement is that the matrix restricted by the function to any finite set of points is positive-semidefinite. Several new kernels can be created by following this criterion. Refer to Christianini and Shawe-Taylor [31] for the detailed proof that if we consider K_1 and K_2 to be kernels over $X \times X, X \subset \mathbb{R}^d$, $a \in \mathbb{R}^+$ where $f(\cdot)$ is a real-valued function on X, and given a linear transformation

$$\phi : X \rightarrow \mathbb{R}^m \tag{37}$$

with K_3 a kernel over $\mathbb{R}^m \times \mathbb{R}^m$ and $\boldsymbol{B}$ a symmetric positive semi-definite $n \times n$ matrix, then the following functions are also valid kernels:

1. $K(\boldsymbol{x}, \boldsymbol{z}) = K_1(\boldsymbol{x}, \boldsymbol{z}) + K_2(\boldsymbol{x}, \boldsymbol{z})$
2. $K(\boldsymbol{x}, \boldsymbol{z}) = \alpha K_1(\boldsymbol{x}, \boldsymbol{z})$
3. $K(\boldsymbol{x}, \boldsymbol{z}) = K_1(\boldsymbol{x}, \boldsymbol{z}) K_2(\boldsymbol{x}, \boldsymbol{z})$
4. $K(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x}) f(\boldsymbol{z})$
5. $K(\boldsymbol{x}, \boldsymbol{z}) = K_3\big(\phi(\boldsymbol{x}), \phi(\boldsymbol{z})\big)$
6. $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}' \boldsymbol{B} \boldsymbol{z}$.

Besides combining existing kernels according to the rules above to make new kernels, kernels can also be created based on the features of the experimental / collected data. The making of new kernels can be accomplished by starting with the features and by determining their inner product. When creating kernels this way, we do no need to verify that the kernel is positive semi-definite, since that condition follows automatically from the inner product definition [31]. For this research, only the linear, polynomial, and RBF kernels are considered in the SVM context. The exploration of custom kernels is left for future research.

The last section of this section deals with the applicability of the SVM approach to the binary classification problem of MCU upset. For mathematical convenience, the outcomes of the MCU upset problem are labeled as +1 denoting upset and 0 denoting no upset. The underlying pattern of this problem could be anything, and no assumption is made regarding X other than it being a set, however in order to apply the theory of SVMs to this problem, additional types of structures are needed. Specifically, in machine learning problems the goal is to be able to generalize the machine learning algorithm to unseen data and have it perform adequately. In the case of binary classification, this means that for some new pattern $x \in X$, we would like to accurately predict the corresponding output labels $y = (1, 0)$. Generally speaking, the output label y will be chosen in such a way that the input-output pair (x, y) is chosen in a similar fashion as represented by the training examples.

Characterizing the similarity of the outputs in the binary case is relatively straightforward: the two labels (upset/no upset) can either be identical or different. The way by which we measure this similarity based on the inputs is the core question that is fundamental to the field of machine learning. To formalize the problem, consider a similarity measure of the form

$$k: X \times X \to \mathbb{R} \tag{38}$$

$$(x, x') \to k(x, x'), \tag{39}$$

which is a function that, given two patterns x and x', returns a real number that characterizes the similarity between the two patterns. The symbol K denotes the kernel function that is assumed to be symmetric [35].

Starting with a simpler notion, the measure of similarity that could apply to this case is the dot product. For instance, given two vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$, a canonical dot product is defined as

$$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle := \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} [\boldsymbol{x}]_{\boldsymbol{i}} [\boldsymbol{x}']_{\boldsymbol{i}}, \tag{40}$$

where $[\boldsymbol{x}]_{\boldsymbol{i}}$ denotes the i^{th}entry of the vector $\boldsymbol{x}$**.** The dot product is also referred to as inner product or the scalar product and sometimes is represented alternatively by $(\boldsymbol{x} \cdot \boldsymbol{x}')$. Geometrically, the dot product can be interpreted as the cosine of the angle between the vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$. Additionally, the geometric representation allows the computation of the length of the vector, also referred to as the norm, i.e.

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}. \tag{41}$$

Moreover, the distance between two vectors is computed as the length of the difference vector. Therefore, the ability to compute the dot product enables us to formulate all the geometric constructions in terms of angles, lengths, and distances. Even though this seems like a simple approach, most interesting problems require a more complex SVM solution.

Having discussed the basic theory and architecture behind SVMs, we now discuss a basic process flow used by SVMs in arriving at classification and regression solutions. Fig. 23 shows a typical architecture of SVMs and related kernel methods. This case assumes that we are dealing with the more complex case of multi-class classification, instead of binary classification, but the concept applies to both. As shown in Fig. 23, the inputs x and the expansion patterns x_i are nonlinearly mapped by $\phi(\cdot)$ into a (possibly infinite) feature space $\mathcal{H}$, also referred to as a higher-dimensional Hilbert-space, in which dot products are computed. As described earlier, these two steps are accomplished in a single step through the use of the kernel k. Next, the results are linearly combined using weights v_i, found by solving a quadratic problem, which in pattern recognition (or binary classification) is represented by $v_i = y_i \alpha_i$ and in regression estimation $v_i = \alpha_i^* - \alpha_i$, or by solving an eigenvalue problem using kernel principal component analysis (PCA). As the final step, the linear combination is processed by the function σ to produce a classifier or regressor value, which in pattern recognition (MCU upset binary classification) is

$\sigma(x) = \text{sgn}(x + b)$, and for completeness, in regression estimation is $\sigma(x) = x + b$, and in kernel PCA is $\sigma(x) = x$ [35].

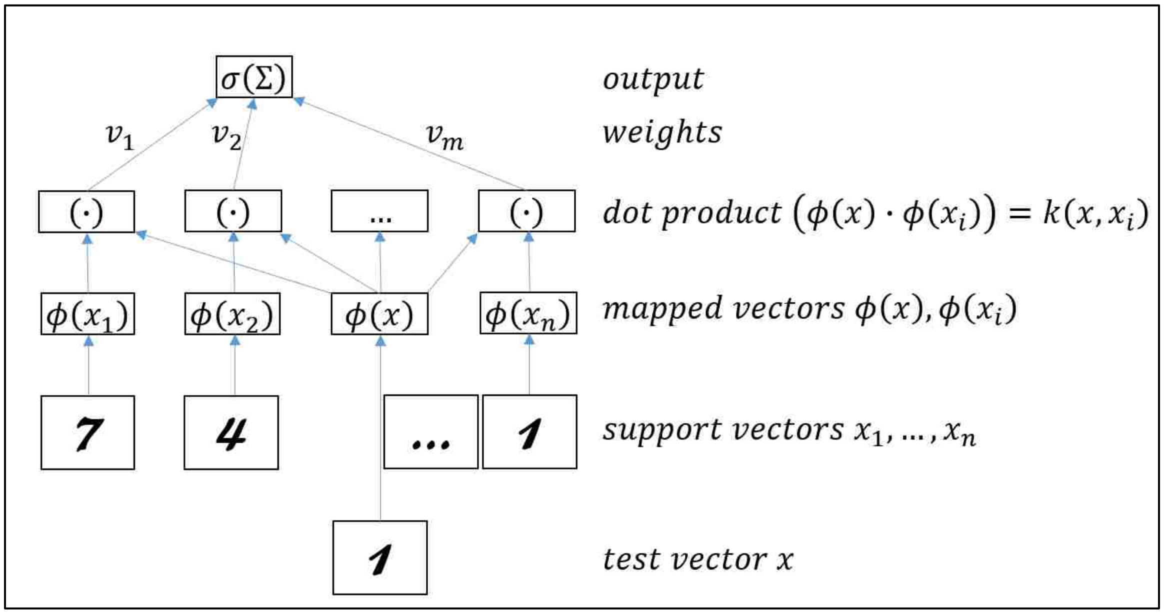


Figure 23: Typical architecture of SVMs and related kernel methods [35].

SVMs for classification and regression require that a kernel function and the parameter C be specified, along with any applicable kernel scale parameters. Typically, C is chosen by k-fold cross-validation. K-fold cross validation uses part of the available data to fit the model (choose C and kernel scale σ) and a different part to test it. During this process, the data is split into K roughly equal-sized parts. For example, suppose that $K = 5$, then the data will be split into five roughly equal parts, as shown in Fig. 24.

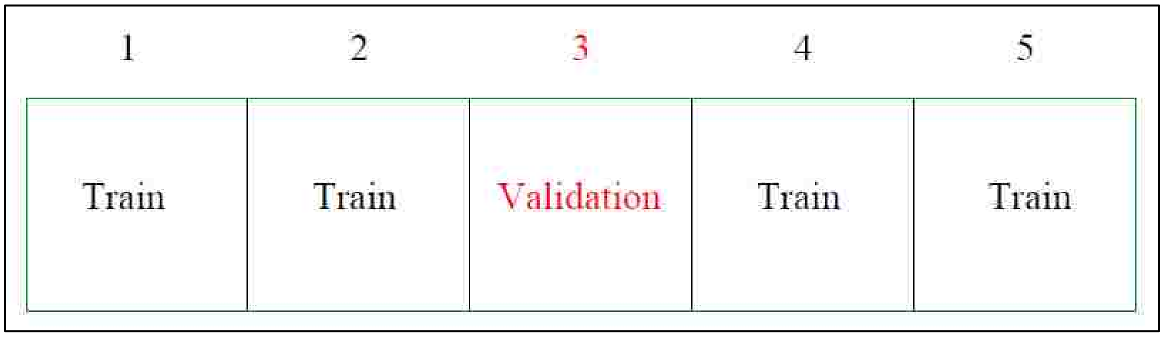


Figure 24: 5-fold cross-validation data split [36].

For the $k-th$ part (data subset 3 above), the algorithm fits the model to the other $K-1$ parts of the data (subsets 1,2,4,5), and calculates the prediction error of the fitted model during the prediction process for the $k-th$ part of the data. The algorithm does this for $k=1,2,3,\ldots,K$ and combines the K estimates of prediction error, usually by averaging them. The cross-validation estimate of the prediction error is given by

$$CV(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, \hat{f}^{-\kappa(i)}(x_i)\right). \tag{42}$$

Typical choices of K are 5 or 10. The case where K equals the number of data points is known as leave-one-out-validation, which is computationally prohibitive for large datasets.

In this research the Bayesian approach is used with cross-validation. As with other types of optimization methods, the goal of the Bayesian approach is to find the extrema of a function $f(x)$ on some bounded set X. In this case, we are looking to choose C and σ in such asway, as to minimize the cross-validation loss in equation 25. What makes the Bayesian approach unique is that it constructs a probabilistic model for $f(x)$ and then exploits this model to make decisions about where in X to next evaluate the function, while integrating out uncertainty. The key idea is to use all of the information available from previous evaluations of $f(x)$ and not simply to rely on local gradient and Hessian approximations. This results in a procedure that can find the minimum of difficult non-convex functions with relatively few evaluations (30 in the MCU upset case) at the cost of performing more computations to determine the next point to evaluate. These computations are usually expensive to perform when training a machine learning algorithm, but the result enables the machine to perform better decisions. MATLAB implements Bayesian optimization with a GP prior, and it gives a range of choices for the so-called acquisition function, which is used to construct a utility function from the model posterior, that is used to determine the next point to evaluate [37].

As introduced earlier, the GP is a convenient and powerful prior distribution on functions of the form $f: X \rightarrow \mathbb{R}$. It is defined by a property in which any finite set of N points induces a multivariate Gaussian distribution on $\mathbb{R}^n$. The $n-th$ of these points is taken to be the function value $f(x_n)$ and the marginalization properties of the Gaussian distribution allow us to compute the marginals and conditionals in closed form. The resulting distributions in functions are determined by a mean function and a covariance function.

Assuming that a function $f(x)$ is drawn from a GP prior and that our observations have the form $\{x_n, y_n\}_{n=1}^{N}$, where $y_n \sim \mathcal{N}(f(x_n), v)$ where v is the variance of the noise in the observations. The prior together with the data induces a posterior over functions. In Bayesian optimization, the acquisition function $a: X \rightarrow \mathbb{R}^{+}$ determines which point in X should be evaluated next via a proxy optimization denoted by $X_{next} = \text{argmax}_X a(X)$ where several different functions have been proposed. These acquisition functions depend on the previous observations as well as on the GP hyperparameters, and this dependence is denoted by $a(x;\ x_n, y_n, \theta)$. There are several popular choices for the acquisition function, but the one that will be utilized to optimize the choice of SVM kernel parameters is a version of the expected improvement acquisition function, given by

$$\begin{aligned} a_{EI}(x;\ x_n, y_n, \theta) \\ = \sigma(x;\ x_n, y_n, \theta)\left(\gamma(x)\Phi\big(\gamma(x)\big) + \mathcal{N}(\gamma(x); 0,1)\right), \end{aligned} \tag{43}$$

where $\gamma(x) = \frac{f(x_{best}) - \mu(x; x_n, y_n, \theta)}{\sigma(x; x_n, y_n, \theta)}$ and $\Phi\big(\gamma(x)\big)$ is the maximized probability of improvement over the current best result.

Chapter 3-3: Artificial Neural Networks (ANN)

One of the computational techniques that will be utilized to model transient upset events in microcontrollers is based on ANNs. Just like SVMs and GPs for machine learning, this technique is based on algorithms that can learn from, and make predictions on data. Rather than utilizing strictly static programming instructions, ANNs build a model from sample data to make predictions or decisions. An important property of NNs is their ability to perform input-output mapping by way of supervised or unsupervised learning. In this learning process, there exists a collection of input-output pairs, which will be used to train a neural network and perform MCU upset modeling tasks. In conjunction with a priori experimental data, the NN should be able to learn which set of input pairs is most likely to cause an upset event. Naturally, the NN may not be able to produce error-free results, but it has a built-in capability to adapt its synaptic weights to changes in the environment in which they operate to minimize the error measure [38].

ANNs are a biologically-inspired model that is enjoying considerable engineering success in applications ranging from time series prediction to computer vision and even electromagnetics as covered in the literature search chapter. ANNs are a good candidate model for the learning from data task because they can efficiently approximate complex target functions. Another reason why they present a suitable tool is that they are usually packaged with good algorithms for data fitting. This section introduces the concept behind ANNs and how to train them on collected data. Moreover, a number of techniques will be presented that fit the data by minimizing the in-sample error. Neural networks are a very flexible model with great approximation power, but they can be prone to overfitting, if not properly utilized. Several techniques that are specific to neural networks are available to control overfitting.

The computational technique that will be utilized to model transient upset events in microcontrollers is ANNs. Fig. 25 shows a summary of the utilization of ANNs to address a data analysis problem.

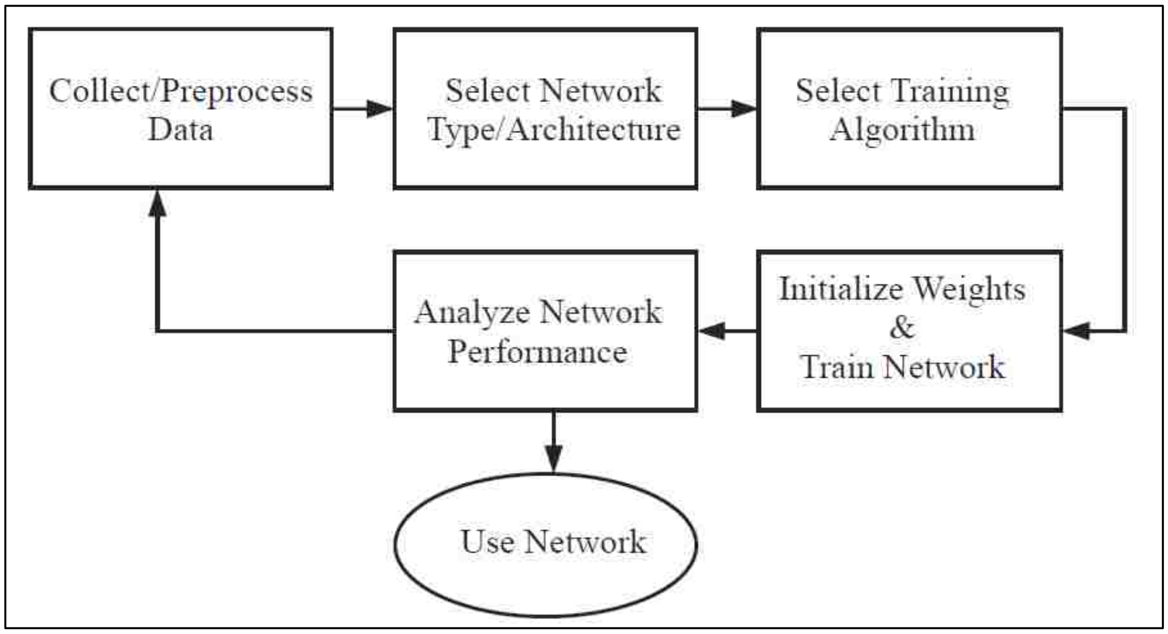

Collect/Preprocess Data
Select Network Type/Architecture
Select Training Algorithm
Initialize Weights & Train Network
Analyze Network Performance
Use Network

Figure 25: Illustration of problem-solving flow using ANNs [38].

The modern perspective on ANNs began in the 1940s with the McCulloch and Pitts neuron. They showed that ANNs could, in principle, compute any arithmetic or logical function. Their work is often acknowledged as the foundation of the ANN field. Following McCulloch and Pitts, Webb posed that classical conditioning, that was discovered by Pavlov is present because of individual neurons' properties, based on which he proposed a mechanism for learning in biological neurons. The next significant development provided for the first application of ANNs came in the late 1950s. At the core of this idea was the invention of the perceptron network and the associated learning rule as proposed by Rosenblatt in 1958. Rosenblatt and his colleagues built a perceptron network and demonstrated that it is able to perform pattern recognition, a success which generated a great deal of interest in ANN research, even though the basic perceptron has been shown to be able to solve only a limited class of problems. Around the same time, Widrow and Hoff introduced a new learning algorithm that is used to train adaptive linear neural networks. These networks were similar in structure and capability to Rosenblatt's perceptron. The Widrow-Hoff learning rule [39] is still used today. However, both Rosenblatt's and Widrow's network suffered inherent limitations, the details of which do not warrant additional coverage here.

Interest in neural networks research all but halted in the late 1960s because of the lack of innovative ideas and the lack of powerful computers that can be used to aid the research. However, these impediments were overcome in the 1980s when the use of personal computers and the introduction of two new concepts reinvigorated the research. The first concept that helped reinvigorate the research into ANNs was the use of statistical mechanics to explain the operation of a certain class of recurrent network. Hence, the class of networks that resulted from this theory was termed Hopfield networks. The second key development was the backpropagation algorithm for training the MLP. It should be noted that several independent researchers discovered this algorithm almost simultaneously. However, the most influential description was published by Rumelhart and McClelland in 1986 [40]. Indeed, many of the advances in ANNs have had to do with innovative architectures and training rules, that along with the availability of powerful new computers continue to drive the research today [38].

The fundamental concept in the field of ANNs is the single input neuron, shown in Fig. 26.

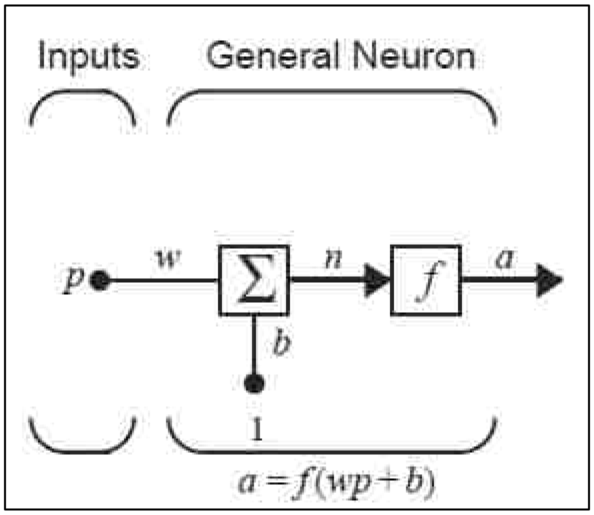


Figure 26: Illustration of a single input neuron [38].

The scalar input p is multiplied by the scalar weight w to form wp, which is one of the terms sent to the adder block. The other input, which is the scalar 1 is multiplied by a bias b and then passed to the same adder block. The output of this adder, labeled n is referred to as the net input, which goes into the transfer function, also referred to as the activation function, f, producing a scalar output a. More formally, the neuron output if calculated as

$$a = f(wp + b), \tag{44}$$

and if the weight, input, and bias is defined as $w = 3$, $p = 2$, and $b = -1.5$, then the neuron output becomes

$$a = f(3(2) - 1.5) = f(4.5). \tag{45}$$

The actual output a depends on the transfer function that is chosen. The terms transfer and activation function refer to the same concept and are used interchangeably. The bias acts like a weight, except that its input is set to a constant of 1, but it can be omitted. In this case, w and b are adjustable scalar parameters of the neuron. The transfer function is chosen by the designer while w and b are adjusted by some learning rule so that the neuron input/output relationship meets a specific goal.

The transfer function f introduced in Fig. 27 may be a linear or nonlinear function of n. Moreover, any particular transfer function is chosen in a way such that it satisfies some specification of the problem that the neuron and the ANN are attempting to solve. The three most commonly used transfer functions are the hard limit, the linear and the log-sigmoid transfer function.

The hard limit transfer function, as shown in Fig. 27 below sets the output of the neuron to 0 if the function argument is less than 0, or it sets the output to 1 if its argument is positive. This transfer function is used to create neurons whose task is to classify inputs into two distinct categories, as applied to the MCU upset problem [38].

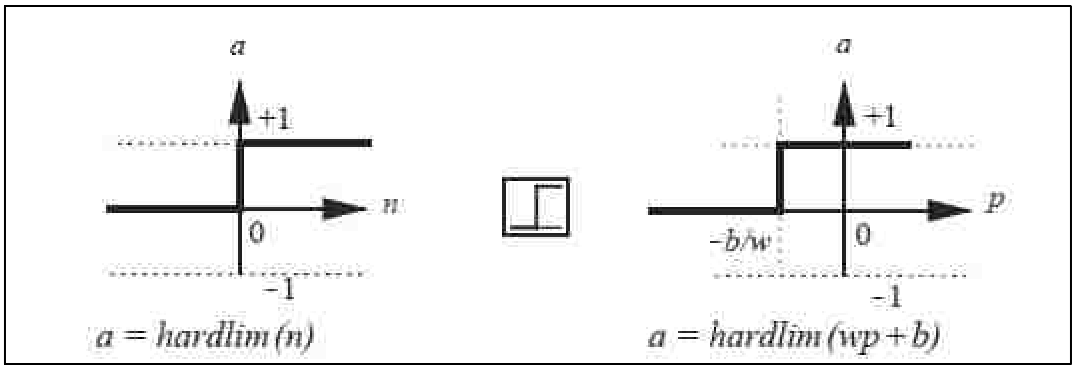


Figure 27: Hard limit transfer function [38].

The second most commonly used transfer function is the linear transfer function:

$$a = n, \tag{46}$$

as shown in Fig. 28.

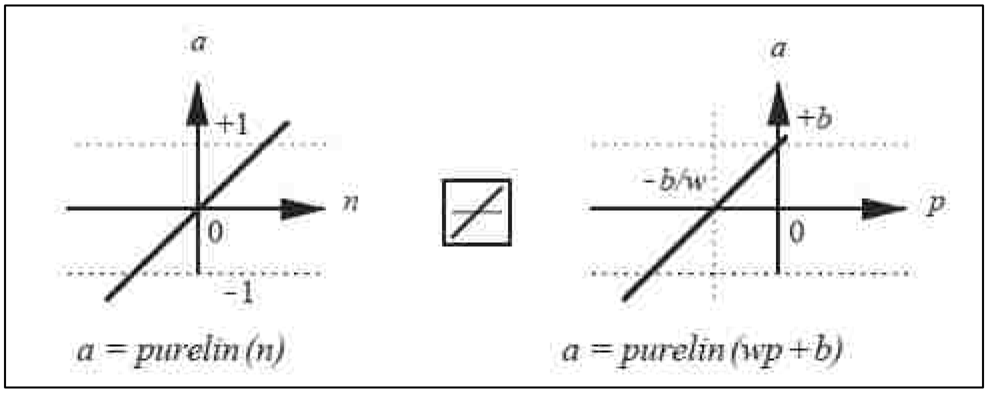


Figure 28: Linear transfer (activation) function [38].

The neuron output a versus input p characteristic is shown in Fig. 28 (right).

Finally, the log-sigmoid activation function is shown in Fig. 29. This function takes the input, which may have a range $-\infty < p < +\infty$ and squashes the output into the range $0 \leq a \leq 1$ according to the following expression:

$$a = \frac{1}{1 + e^{-n}}. \tag{47}$$

The log-sigmoid activation function is commonly used in multi-layer ANNs that are trained using the backpropagation algorithm because this function is differentiable. Other commonly used transfer functions are summarized in Appendix A [38].

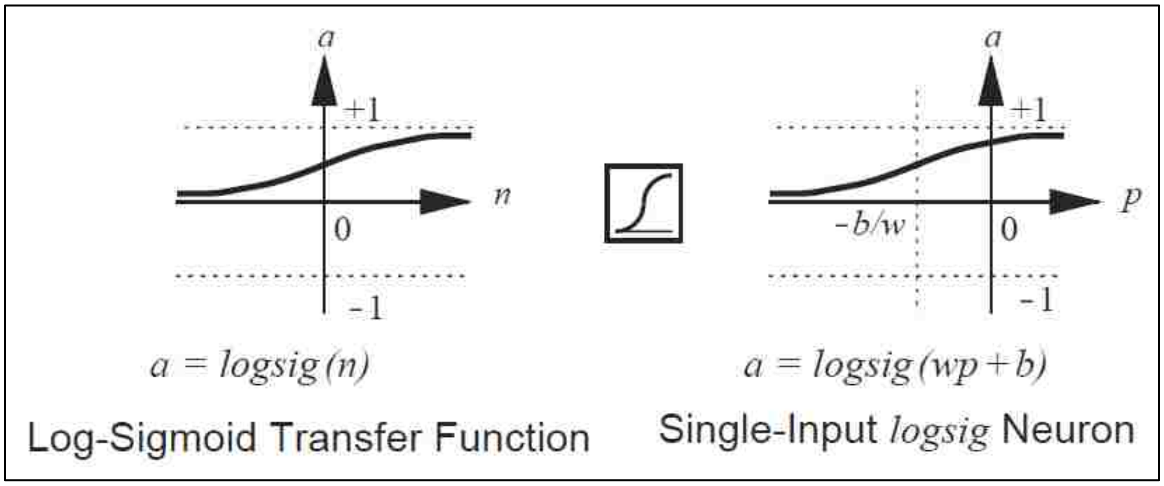


Figure 29: Sigmoid transfer (activation) function [38].

While a single input neuron is used to illustrate the general concept, typically in practice a neuron has multiple inputs, each of which corresponds to one dimension of the input data. Such a neuron is shown in Fig. 30. In this case, the individual inputs $\boldsymbol{p} = \{p_1, \ldots, p_n\}$ are each weighted by the corresponding weight elements $\boldsymbol{W} = \{\boldsymbol{w_{1,1}}, \ldots, \boldsymbol{w_{1,R}}\}$ of the weight matrix. The neuron has a bias b that is summed with the weighted inputs resulting in $n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b$ or in matrix form $n = \boldsymbol{Wp} + b$ in which the matrix $\boldsymbol{W}$ is a single row for the single neuron case. The neuron output can now be written as [38]

$$a = f(\boldsymbol{Wp} + b). \tag{48}$$

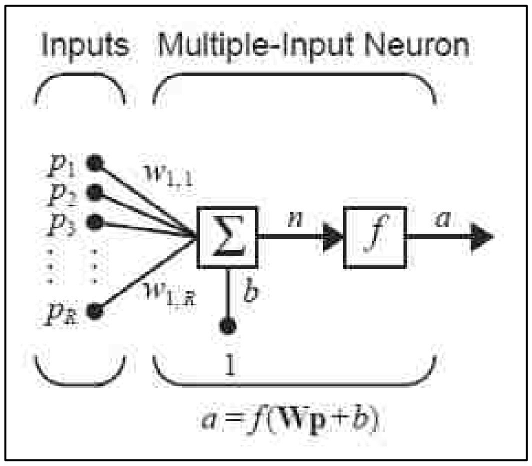


Figure 30: Multiple input neuron [38].

The issue with the single perceptron or neuron is that it cannot implement simple classification functions. Fig. 31 illustrates a scenario in which a single perceptron is ill-suited to find a solution. The example in Fig. 31 is related to the Boolean XOR function. In this case, the classifying function f cannot be written as $\text{sign}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})$, because it is composed of two linear parts. To solve the XOR problem, the classification function f can be decomposed into two simple perceptrons corresponding to the two lines separating the two classes. The outputs of these two perceptrons can be combined to reconstruct the classification function f. Fig. 32 shows the two perceptrons required to successfully tackle the XOR problem.

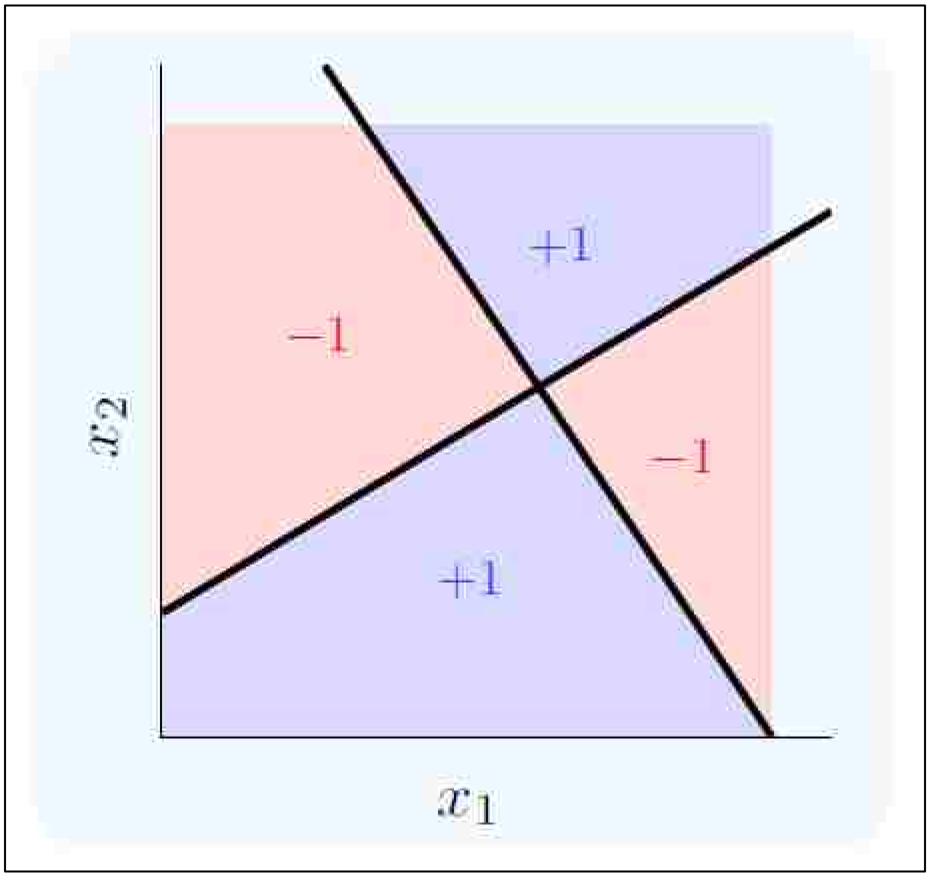
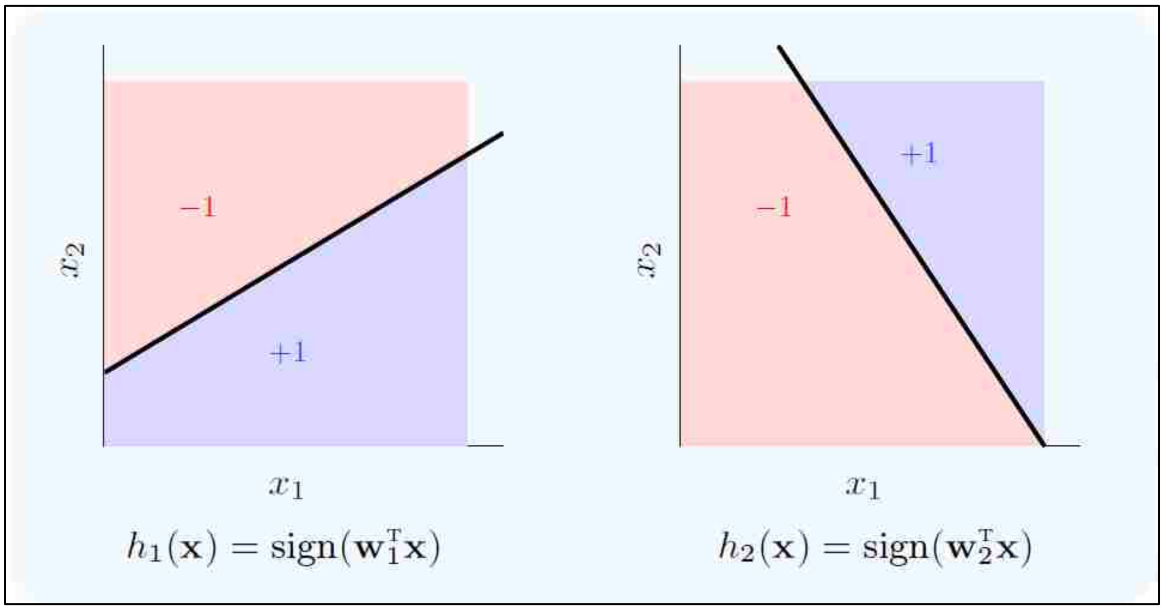


Figure 31: Classification scenario ill-suited for single perceptron [29].



-1
$+1$
x_2
x_1
$h_1(\mathbf{x}) = \text{sign}(\mathbf{w}_1^\mathsf{T}\mathbf{x})$
$+1$
-1
x_2
x_1
$h_2(\mathbf{x}) = \text{sign}(\mathbf{w}_2^\mathsf{T}\mathbf{x})$

Figure 32: Two perceptrons required to solve the XOR problem illustrated in Fig. 23 [29].

The classifying target function f equals $+1$ when exactly one of h_1, h_2 equals $+1$. Using standard Boolean notation, this target function can be written as

$$f = h_1\overline{h_2} + \overline{h_1}h_2, \tag{49}$$

implying that more complicated targets (problems) can be solved by various combinations of perceptrons as shown in Fig. 33.

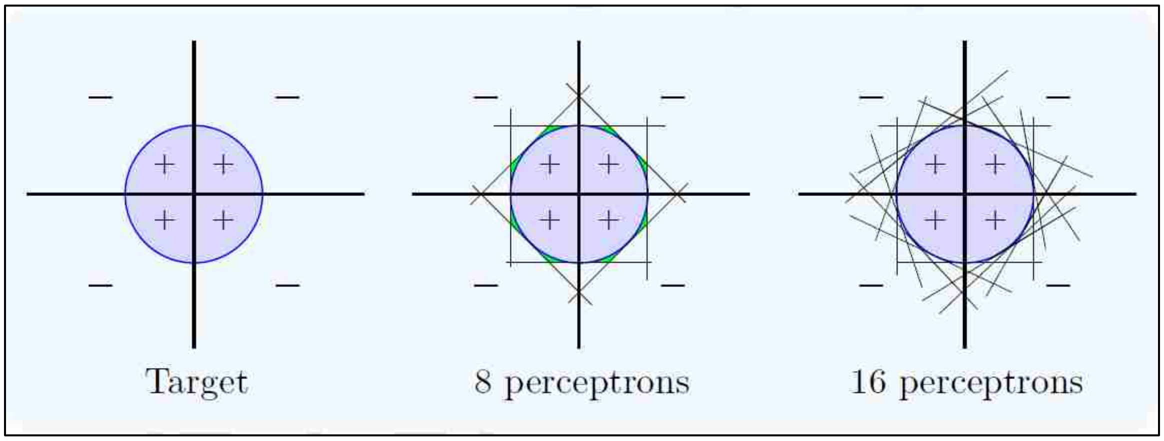


Figure 33: Target classification with different numbers of perceptrons [29].

The formal proof of this concept is somewhat similar to the theorem of calculus which says that any continuous function on a compact set can be approximated arbitrarily closely using step functions. In the scenario of Fig. 33, the perceptron is the analog of the step function [29]. Thus, more complex target functions can be modeled by adding more nodes (hidden units) in the hidden layers, corresponding to allowing more perceptrons to work on the decomposition of the target function f, which we wish to learn.

In practice, a single neuron with many inputs may not be sufficient. It may be necessary to employ five or ten, maybe more layers that are operating in parallel, forming a neuron layer. A single-layer network of S neurons is shown in Fig. 34.

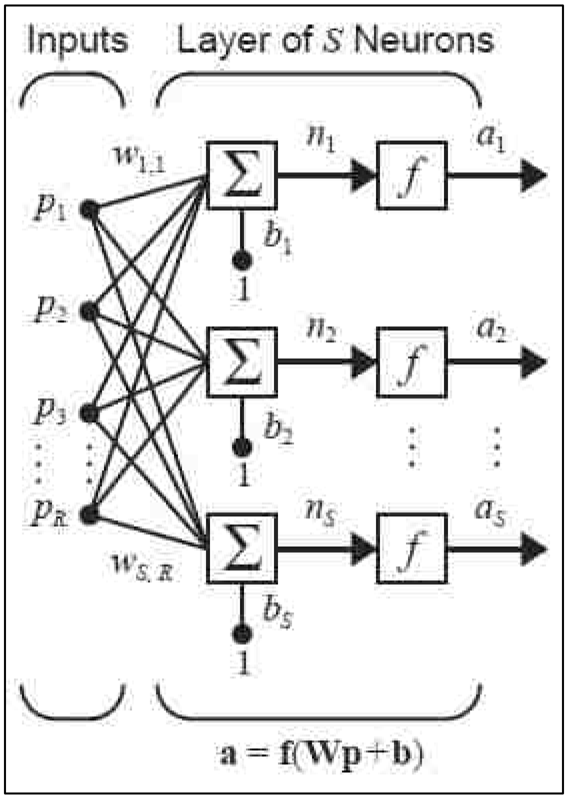


Figure 34: A layer of S neurons [38].

For this case, each of the R inputs is connected to each neuron, resulting in a weight matrix $\boldsymbol{W}$ with S rows. The layer includes the weight matrix, the summers, the bias vector $\boldsymbol{b}$, the transfer function boxes, and finally the output vector a. Some literature refers to the input layer as a separate layer. Each element of the input matrix $\boldsymbol{p}$ is connected to each neuron through the matrix $\boldsymbol{W}$, defined as

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix} \tag{50}$$

where the row indices indicate the destination neuron associated with that weight, and where the column indices indicate the source input for that weight. For example, the indices in $w_{1,2}$ indicate that this weight represents the connection to the first neuron from the second source. Fig. 35 shows the MLP drawn in abbreviated notation [38].

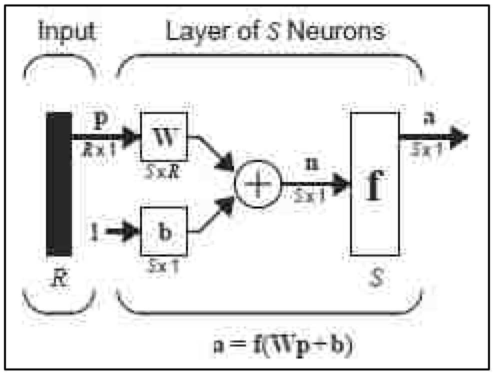


Figure 35: Layer of S neurons in abbreviated notation [38].

Similarly, an ANN can contain multiple layers. This type of configuration is referred to as the MLP described earlier, and is shown in Fig. 36.

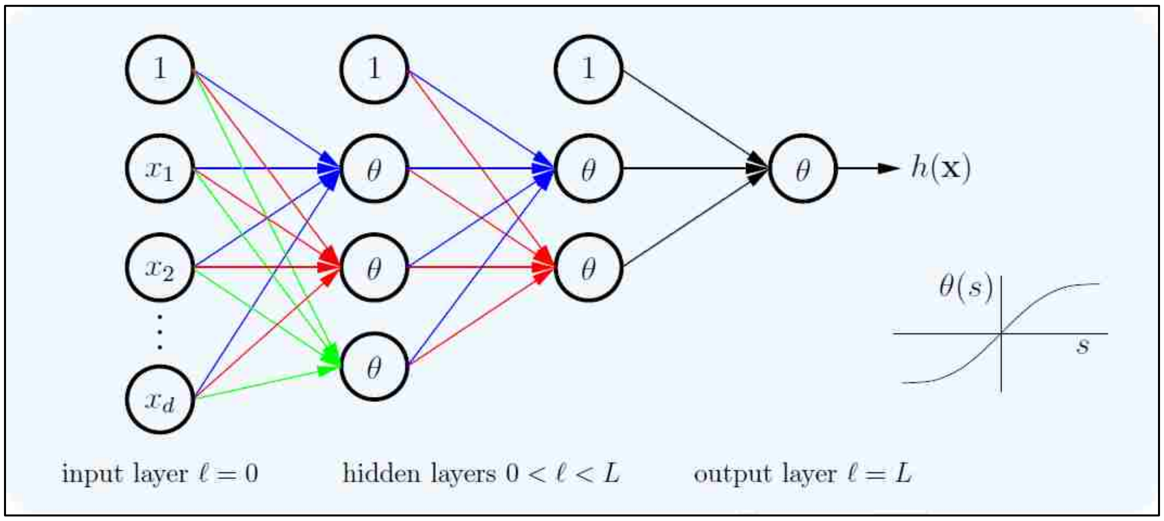


Figure 36: Illustration of a 'softened' MLP ANN [29].

The ANN in Fig. 36 is an example of a feed-forward MLP. To facilitate concise mathematical description, a slight change in notation follows. The layers are labeled by $l = 0,1,2, \dots L$. Fig. 36 shows a ANN with $L = 3$, meaning that there are three layers, the input layer, the hidden layer, and the output layer, even though the input layer is usually not considered a layer in the traditional sense. The layer L is the output layer and, it determines the value of the function. The layers $0 < l < L$ are the hidden layers. The information flows from the inputs on the far left, along links and through the hidden nodes, to the output $h(\boldsymbol{x})$ on the far right [29].

The $\text{superscript}^{(l)}$ is used to refer to a layer, each of which has dimension $d^{(l)}$. This means that each layer has $d^{(l)} + 1$ nodes, labeled $0,1, \dots, d^{(l)}$. As mentioned earlier, each layer has a special node, also called a bias node. The bias node is set to have an output value of 1. In an ANN every arrow represents a weight or connection strength from one node in a layer to a node in the next higher layer. Generally, if a node has an incoming arrow that indicates that this node has an incoming signal. Every node with such an input signal has a transformation or transfer function θ. In the case of MLP for classification, this transfer function is $\theta(s) = \text{sign}(s)$. A soft version of the MLP with $\theta(x) = \tanh(x)$ can be used to approximate the $\sinh(\cdot)$ function. The $\tanh(\cdot)$ function is a soft threshold or sigmoid, whose output, in the case of classification, is replaced by the hard threshold $\text{sign}(\cdot)$ [29].

The ANN architecture is what determines the neural network model $\mathcal{H}_{nn}$. The architecture of the network is determined by the dimension of each layer $\boldsymbol{d} = \left[d^{(0)}, d^{(1)}, \dots, d^{(L)}\right]$, where L represents the number of layers in the ANN. The determination of the interconnect weights specifies a hypothesis $h \in \mathcal{H}_{nn}$. For example, consider a node in a hidden layer shown in Fig. 37.

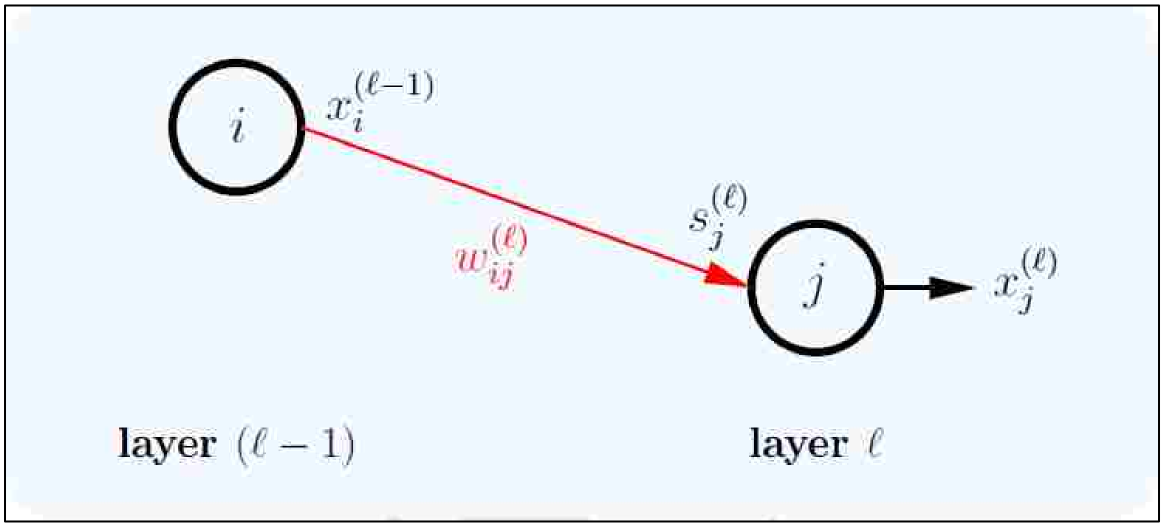


Figure 37: Example of an ANN node in a hidden layer [29].

Consider a node that has an incoming signal s and an output x. A pair of incoming signals and outgoing signals forms a link between two nodes that has a weight $w^{(l)}$ associated with it. These weights are indexed by the layer for which they act as input signals, therefore the output of the nodes in layer $l-1$ is multiplied by the weights of the links that act as input signals to layer l, or $w^{(l)}$. Subscripts on the weights are used to index the nodes in a layer, meaning that $w_{ij}^{(l)}$is the weight into node j in layer l from node i in the previous layer. The signal $s_j^{(l)}$ denotes the signal going into node j in the layer l. The output of the node j is denoted as $x_j^{(l)}$. The ANN also contains some special nodes. The zero nodes in every layer are called constant nodes (bias nodes) and each of their outputs is set to 1. These nodes do not have an incoming weight, but they do have an outgoing weight. The nodes in the input layer ($l = 0$) are reserved for the input values and they have no incoming weight or transformation function [29].

Generally, the theoretical treatment of ANNs is dealt with on a layer-by-layer basis. All the input signals to nodes $1, \ldots, d^{(l)}$ and all the output nodes $0, \ldots, d^{(l)}$ in the layer l are collected in the vector $\boldsymbol{x} \in \{1\} \times \mathbb{R}^{\boldsymbol{d}^{(l)}}$. Additionally, there are links that connect the outputs of all nodes in a previous layer to the inputs of layer l. Therefore, a $\left(d^{(l-1)} + 1\right) \times d^{(l)}$ matrix of weights $W^{(l)}$ serves as input to layer l. The i, j entry of $W^{(l)}$ is $w_{ij}^{(l)}$ going from node i in the previous layer to node j in layer l. A concise summary of this concept is illustrated in Fig. 38 and its accompanying Table 7 [29].

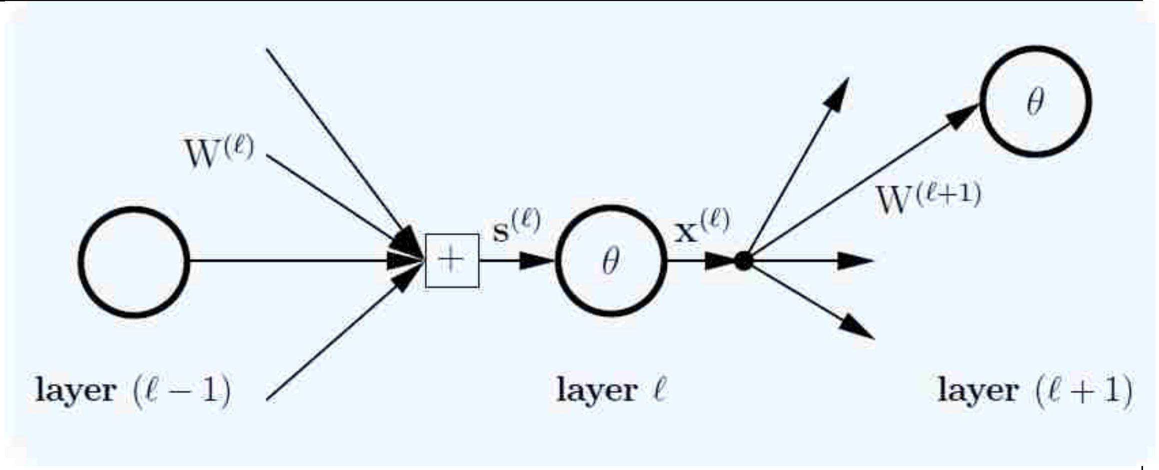


Figure 38: Example of an ANN node [29].

Table 7: layer l parameters of ANN node of Fig. 27

signals in	$\boldsymbol{s}^{(l)}$	$d^{(l)}$ dimensional input vector
outputs	$\boldsymbol{x}^{(l)}$	$d^{(l)} + 1$ dimensional output vector
weights in	$\boldsymbol{W}^{(l)}$	$\left(d^{(l-1)} + 1\right) \times d^{(l)}$ dimensional matrix
weights out	$\boldsymbol{W}^{(l+1)}$	$\left(d^{(l)} + 1\right) \times d^{(l+1)}$ dimensional matrix

Once the weights $W^{(l)}$ for $l = 1, \dots, L$ are determined, we have specified a neural network hypothesis $h \in \mathcal{H}_{nn}$. All the weight matrices are collected into a single weight parameter $\boldsymbol{w} = \left\{W^{(1)}, W^{(2)}, \dots, W^{(L)}\right\}$, sometimes written in short notation as $h(\boldsymbol{x}; \boldsymbol{w})$ indicating the dependence of the hypothesis on $\boldsymbol{w}$ [29].

The neural network hypothesis is computed via the forward propagation algorithm. The input-output relationship of each layer is given by

$$\boldsymbol{x}^{(l)} = \begin{bmatrix} 1 \\ \theta\left(\boldsymbol{s}^{(l)}\right) \end{bmatrix} \tag{51}$$

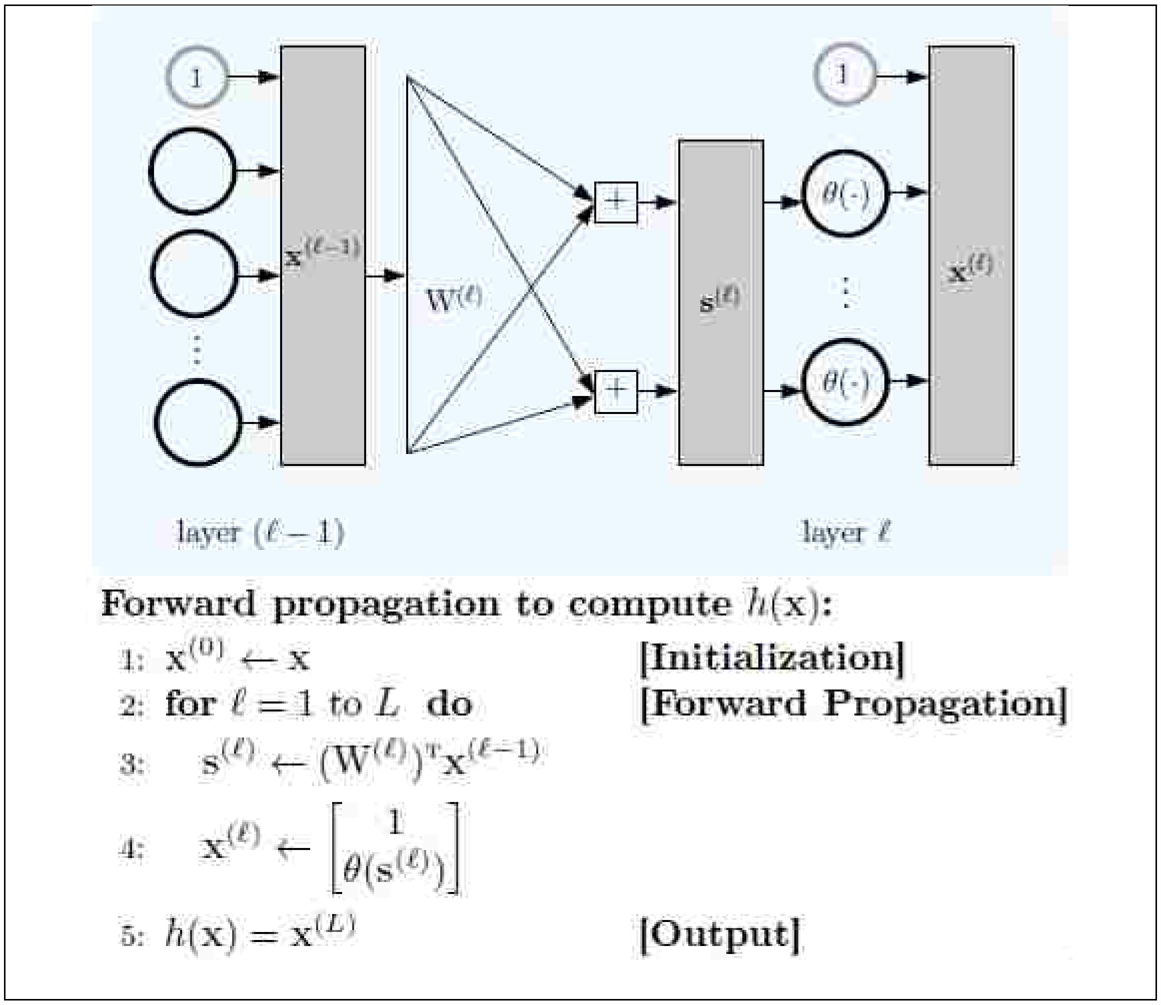
where $\theta\left(\boldsymbol{s}^{(l)}\right)$ is a vector with components $\theta\left(s^{(l)}\right)$. To get the input vector to layer l, the weighted sum of the outputs from the previous layer must be computed. The weights are specified by

$$\boldsymbol{s}^{(l)} = \left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{x}^{(l-1)}. \tag{52}$$

The remaining step is to initialize the input layer to $\boldsymbol{x}^{(0)} = \boldsymbol{x}$, so $d^{(0)} = d$, which is the input dimension and use the equations for $\boldsymbol{x}^{(l)}$ and $\boldsymbol{s}^{(l)}$ introduced above in the following chain

$$\boldsymbol{x} = \boldsymbol{x}^{(0)} \xrightarrow{\boldsymbol{W}^{(1)}} \boldsymbol{s}^{(1)} \xrightarrow{\theta} \boldsymbol{x}^{(1)} \xrightarrow{\boldsymbol{W}^{(2)}} \boldsymbol{s}^{(2)} \xrightarrow{\theta} \boldsymbol{x}^{(2)} \dots \boldsymbol{s}^{(L)} \xrightarrow{\theta} \boldsymbol{x}^{(L)} = h(\boldsymbol{x}) \tag{53}$$

in which the input vectors are augmented with $\boldsymbol{x}_0 = 1$. The forward propagation algorithm is summarized and illustrated in Fig. 39 [29].



Forward propagation to compute $h(\mathrm{x})$:

```
1: $\mathrm{x}^{(0)} \leftarrow \mathrm{x}$                                    [Initialization]
2: for $\ell = 1$ to $L$ do                              [Forward Propagation]
3:     $\mathrm{s}^{(\ell)} \leftarrow (\mathrm{W}^{(\ell)})^{\mathrm{T}}\mathrm{x}^{(\ell-1)}$
4:     $\mathrm{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathrm{s}^{(\ell)}) \end{bmatrix}$
5: $h(\mathrm{x}) = \mathrm{x}^{(L)}$                                   [Output]
```

Figure 39: Forward propagation algorithm [29].

Once the forward propagation algorithm has completed processing, the computation of the output vector $x^{(l)}$ at every layer $l = 0,1, \dots, L$ has been accomplished. To compute the input error discussed earlier, all that is required are $h(x_n)$ and y_n, with the sum of squares

$$E_{in}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n; w) - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(x_n^{(L)} - y_n\right)^2 . \qquad (54)$$

The weights are learned by minimizing E_{in} by directly applying an efficient gradient descent algorithm [29].

The gradient descent is a general technique that is used to minimize a twice-differentiable function, such as $E_{\text{in}}(\boldsymbol{w})$. In this case, $E_{\text{in}}(\boldsymbol{w})$ is an error surface in high-dimensional space. During the initial step of the gradient descent algorithm, we start somewhere on this surface and attempt to find a local minimum, thereby decreasing the in-sample error $E_{\text{in}}(\boldsymbol{w})$ in the case of logistic regression problems, however it is possible to use it for classification problems as well. The weights are initialized to $\boldsymbol{w}(0)$ and the weights are updated by taking a step in the negative gradient direction

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta \nabla E_{\text{in}}\big(\boldsymbol{w}(t)\big), \text{for } t = 1,2,3,\dots \tag{55}$$

the implementation of which requires the gradient.

Consider a sigmoidal multi-layer network with $\theta(x) = \tanh(x)$. Since $\tanh(x)$ and thus $h(\boldsymbol{x})$ is a smooth function, the gradient descent algorithm can be applied to the resulting error function. To apply this algorithm, the gradient $\nabla E_{\text{in}}(\boldsymbol{w})$ is needed, where the weight vector $\boldsymbol{w} = W^{(1)}, \dots, W^{(L)}$, along with the derivatives with respect to all these weights. For the multilayer sigmoidal network, there exists no simple closed form expression for the gradient. Additionally, consider an in-sample error defined as the sum of point-wise errors over data points

$$E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} e_n, \tag{56}$$

where $e_n = e(h(x_n), y_n)$, which in case of the squared error measure is defined as $e(h, y) = (h - y)^2$. To compute the gradient of E_{in}, the derivative with respect to each weight matrix is required:

$$\frac{\partial E_{\text{in}}}{\partial W^{(l)}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial e_n}{\partial W^{(l)}}, \tag{57}$$

which can be determined via the finite numerical difference approach. Backpropagation is based on several applications of the chain rule to express partial derivatives in layer l using partial derivatives in layer $(l+1)$. The backpropagation algorithm is based on the sensitivity vector for layer l, denoted as $\boldsymbol{\delta}^{(l)}$and defined as the gradient of the error e with respect to the input signal to the layer l, $\boldsymbol{s}^{(l)}$. The sensitivity vector is defined as

$$\boldsymbol{\delta}^{(l)} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{s}^{(l)}} \tag{58}$$

and it describes how the error changes with $\boldsymbol{s}^{(l)}$. The partial derivatives with respect to the weights $W^{(l)}$ can now be written as

$$\frac{\partial e}{\partial W^{(l)}} = \boldsymbol{x}^{(l-1)}\left(\delta^{(l)}\right)^{T}. \tag{59}$$

The partial derivatives of the weights on the left-hand side form a matrix with dimensions $\left(d^{(l-1)}+1\right) \times d^{(l)}$. The partial derivatives are composed of two components: (i) the sensitivity vector of the layer from which the weights originate, and (ii) the sensitivity vector of the layer into which the weights go. The sensitivity of the error e is directly proportional to the magnitude of the sensitivity vector [29].

As mentioned earlier, the outputs $\boldsymbol{x}^{(l)}$ for every layer are computed by forward propagation and to obtain partial derivatives, it is necessary and sufficient to compute the sensitivity vectors $\boldsymbol{\delta}^{(l)}$ for every layer, which can be accomplished by running a modified version of the network backwards, hence the label backpropagation. More succinctly, in forward propagation, the algorithm computes $\boldsymbol{x}^{(l)}$ from $\boldsymbol{x}^{(l-1)}$and in backpropagation, the algorithm computes $\boldsymbol{\delta}^{(l)}$ from $\boldsymbol{\delta}^{(l+1)}$**.** Thus, in forward propagation, the output of each layer is the vector $\boldsymbol{x}^{(l)}$and in backpropagation the output of each layer is $\boldsymbol{\delta}^{(l)}$, as illustrated in Fig. 40.

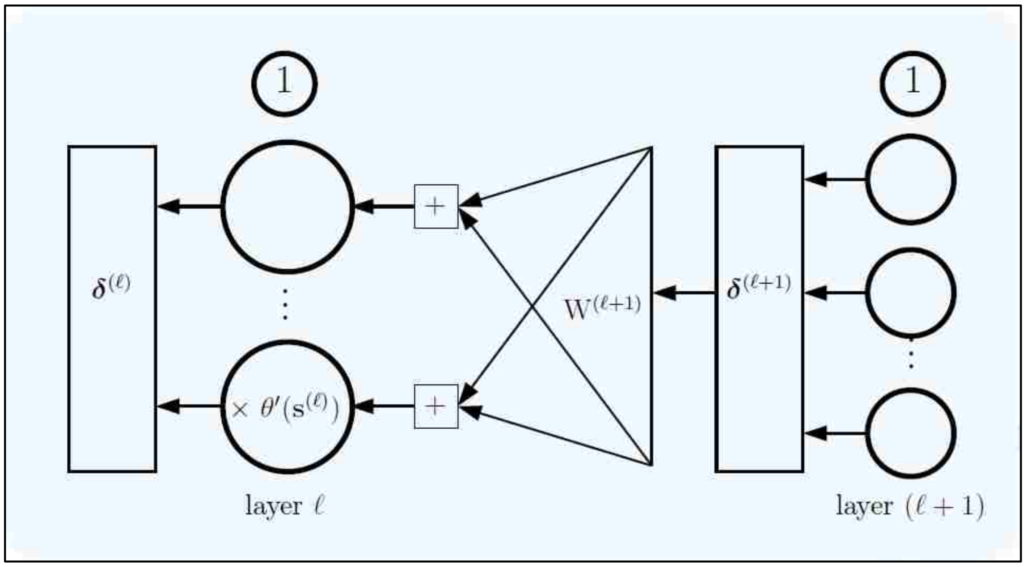
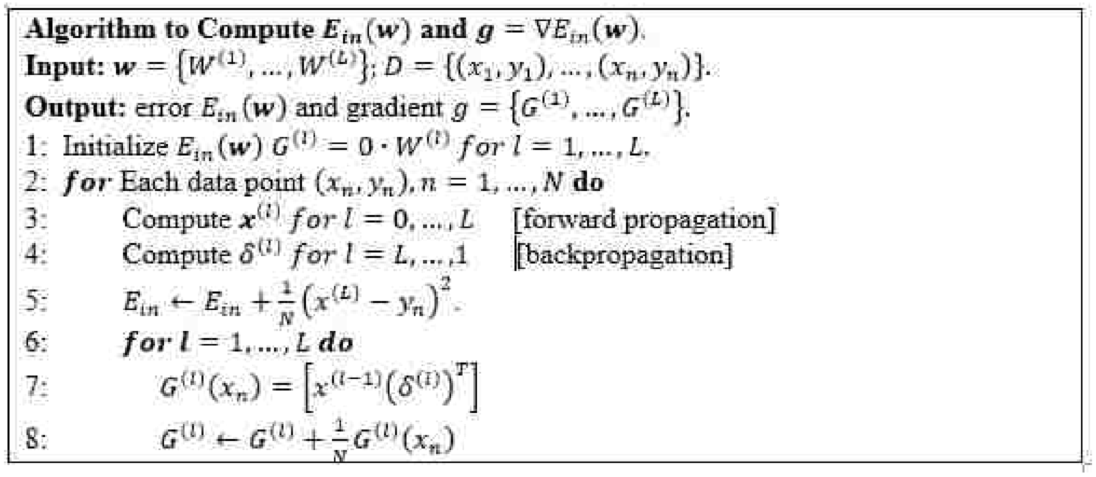


Figure 40: Illustration of the backpropagation process [29].

The network is slightly modified for the backpropagation case. The modification entails a change in the transformation function for the nodes. In forward propagation, the transformation is the sigmoid $\theta(\cdot)$, whereas for the backpropagation case the transformation is the multiplication by $\theta'(\boldsymbol{s}^{(l)})$ where $\boldsymbol{s}^{(l)}$ is the input to the node. The transformation function is now different for each node and it depends on the input to the node, which in turn depends on the input $\boldsymbol{x}$**,** which was computed during the forward propagation. The mathematical notation for the forward propagation case with the $tanh$ transformation function is $\tanh'^{(\boldsymbol{s}^{(l)})} = 1 - \tanh^2(s^{(l)}) = 1 - \boldsymbol{x}^{(l)} \otimes x^{(l)}$, where $\otimes$

denotes component-wise multiplication, whereas the notation for the backpropagation case is $\delta^{(l)} = \theta'\left(s^{(l)}\right) \otimes \left[W^{(l+1)}\delta^{(l+1)}\right]_1^{d^{(l)}}$, where the vector $\left[W^{(l+1)}\delta^{(l+1)}\right]_1^{d^{(l)}}$ contains components $1, \dots, d^{(l)}$ of the vector $W^{(l+1)}\delta^{(l+1)}$ [29].

The backpropagation algorithm works with a data point (x, y) and weights $\boldsymbol{w} = \left\{W^{(1)}, \dots, W^{(L)}\right\}$. One run of the forward and backward propagation algorithms computes the outputs $\boldsymbol{x}^{(l)}$ and the sensitivities $\delta^{(l)}$with algorithm run time that is in order of the number of weights in the network. Generally, this is computed once and for each data point (x, y) to get $\nabla E_{\text{in}}(\boldsymbol{x}_{\text{in}})$ and eventually the full batch gradient $\nabla E_{\text{in}}(\boldsymbol{w})$ defined earlier. The full algorithm is summarized in Fig. 41**.**



```
Algorithm to Compute $E_{in}(w)$ and $g = \nabla E_{in}(w)$.
Input: $w = \{W^{(1)}, \dots, W^{(L)}\}$; $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$.
Output: error $E_{in}(w)$ and gradient $g = \{G^{(1)}, \dots, G^{(L)}\}$.
1: Initialize $E_{in}(w)$ $G^{(l)} = 0 \cdot W^{(l)}$ $for$ $l = 1, \dots, L$.
2: $for$ Each data point $(x_n, y_n), n = 1, \dots, N$ do
3:     Compute $x^{(l)}$ $for$ $l = 0, \dots, L$    [forward propagation]
4:     Compute $\delta^{(l)}$ $for$ $l = L, \dots, 1$    [backpropagation]
5:     $E_{in} \leftarrow E_{in} + \frac{1}{N}\left(x^{(L)} - y_n\right)^2$.
6:     $for$ $l = 1, \dots, L$ $do$
7:         $G^{(l)}(x_n) = \left[x^{(l-1)}\left(\delta^{(l)}\right)^T\right]$
8:         $G^{(l)} \leftarrow G^{(l)} + \frac{1}{N}G^{(l)}(x_n)$
```

Figure 41: The ANN algorithm [29].

In addition to the standard backpropagation algorithms, there are several alternative adaptive learning algorithms for feedforward neural networks that have been developed with the most pertinent one being the scaled conjugate gradient (SCG) for fast supervised learning, originally proposed by proposed by Moller in 1993 [41]. The SCG algorithm for training a feedforward neural network is implemented in the MATLAB neural networks toolbox and will be described in more detail. Standard backpropagation algorithms usually have poor convergence rates, and they depend on user-defined parameters such as learning rate and momentum constant. The additional disadvantage of using the standard backpropagation algorithms is that there is no theoretical backing for an optimal choice of these user-defined parameters [41].

From an optimization perspective, the learning problem in the neural network approach is equivalent to minimizing a global error function, which is a form of a multivariate function that depends on the weights of the network. The approach of minimizing a function is well known in the conventional numerical analysis field. It is the general opinion in the numerical analysis community that the Conjugate Gradient Methods are particularly well suited to tackle large-scale problems in an effective manner. Most of the optimization methods that are used to minimize functions are based on the same strategy. The minimization is a local iterative process in during which an approximation to the function in a neighborhood of the current point in weight space is minimized. This approximation is often given by a first order Taylor expansion of the function. This approach is illustrated in the following the pseudo-algorithm:

1. Choose the initial weight vector $\boldsymbol{w}_1$ and set $k = 1$
2. Determine a search direction $\boldsymbol{p}_k$ and a step size α_k so that $E(\boldsymbol{w_k} + \alpha_k \boldsymbol{p_k}) < E(\boldsymbol{w}_k)$
3. Update vector: $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{p}_k$
4. If $E'(\boldsymbol{w}_k) \neq 0$ set $k = k + 1$ and return to step 2, otherwise return $\boldsymbol{w}_{k+1}$ as the desired minimum.

The determination of the next current point in this iterative process involves two independent steps. During the first step, a search direction is determined, that is in which direction we should move next to determine the minimum value of the error function. In other words, in what direction in the weight space do we need to go next in the search for a new current point. Once the direction has been determined, the next step is to determine how far the algorithm should go in that direction, i.e. a step size needs to be chosen. In the case when the search direction in the above algorithm is set to the negative gradient $-E'(\boldsymbol{w})$ and the step size is set to a constant ε then the algorithm becomes a gradient descent algorithm [41].

Conjugate direction methods also utilize the same general optimization strategy, but they make use of information from the second order approximation $E_{qw}(\boldsymbol{w}+\boldsymbol{y}) \approx E(\boldsymbol{w}) + E'(\boldsymbol{w})^T\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T E''(\boldsymbol{w})\boldsymbol{y}$ to choose the search direction and step size more carefully. Quadratic functions have certain advantages over general functions. Consider the quadratic approximation to the error function E in a neighborhood of a point $\boldsymbol{w}$ given by

$$E_{qw}(\boldsymbol{y}) = E(\boldsymbol{w}) + E'(\boldsymbol{w})^T\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T E''^{(w)}\boldsymbol{y}. \qquad (60)$$

To determine the minima of E_{qw}, we must solve the following linear system to find the critical points

$$E'_{qw}(\boldsymbol{y}) = E''(\boldsymbol{w})\boldsymbol{y} + E'(\boldsymbol{w}) = 0. \qquad (61)$$

If a conjugate system is available, the solution of this linear system is simplified considerably [41]. If and only if the Hessian matrix $E''(\boldsymbol{w})$ is positive definite, then $E_{qw}(\boldsymbol{y})$ has a unique global minimum, otherwise the minimum can be a saddle point or a maximum.

The following two theorems, when combined, are the driving force behind the conjugate gradient algorithm.

THEOREM 1: *Let* $\boldsymbol{p}_1, \dots, \boldsymbol{p}_N$ *be a conjugate system and* $\boldsymbol{y}_1$ *a point in weight-space. Let the points* $\boldsymbol{y}_2, \dots, \boldsymbol{y}_{N+!}$ *be recursively defined by* $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k \boldsymbol{p}_k$ *where* $\boldsymbol{\alpha}_k = \boldsymbol{\mu}_k / \boldsymbol{\delta}_k$, $\boldsymbol{\mu}_k = -\boldsymbol{p}_k^T E'_{qw}(\boldsymbol{y}_k), \boldsymbol{\delta}_k = \boldsymbol{p}_k^T E''(\boldsymbol{w}) \boldsymbol{p}_k$. *Then* $\boldsymbol{y}_{k+1}$ *minimizes* E_{qw} *restricted to the k-plane* π_k *given by* $\boldsymbol{y}_1$ *and* $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ [41].

The algorithm portion that is based on this theory ensures that the global minimum of a quadratic function is detected in, at most, N iterations. Additionally, if the all the eigenvalues of the Hessian $E''(\boldsymbol{w})$ fall into multiple groups of the same size then there is increased probability that the algorithm will terminate in much less than N iterations [41].

To reach the conjugate part of the algorithm, Theorem 2 is used to obtain the conjugate vectors that are also referred to as conjugate gradient directions.

THEOREM 2: *Let* $\boldsymbol{y}_1$*be a point in weight space and* $\boldsymbol{p}_1$ *and let* $\boldsymbol{r}_1$ *equal to the steepest descent vector* $-E'_{qw}(\boldsymbol{y}_1)$. *Additionally, let* $\boldsymbol{p}_{k+1}$ *be recursively defined as*

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{\boldsymbol{k+1}} + \boldsymbol{\beta}_{\boldsymbol{k}} \boldsymbol{p}_{\boldsymbol{k}} \tag{62}$$

where $\boldsymbol{r}_{k+1} = E'_{qw}(\boldsymbol{y}_{k+1}), \boldsymbol{\beta}_k = (|\boldsymbol{r}_{k+1}|^2 - \boldsymbol{r}_{k+1}^T \boldsymbol{r}_k) / \boldsymbol{p}_k^T \boldsymbol{r}_k$ *and* $\boldsymbol{y}_{k+1}$ *is the point generated in Theorem 1. Then* $\boldsymbol{p}_{k+1}$ *is the steepest descent vector to* E_{qw} *restricted to the* $(N-1)$*-plane* π_{N-k} *conjugate to* π_k *given by* $\boldsymbol{y}_1$ *and* $\boldsymbol{p}_1, \dots, \boldsymbol{p}_k$ [41].

The combination of Theorem 1 and Theorem 2 forms the basis of the conjugate gradient algorithm. In each iteration, this algorithm is applied to the quadratic approximation E_{qw} of the global error function E at the current point $\boldsymbol{w}$ in weight space and it is restarted if it does not converge in N iterations. It should be noted that this algorithm is valid only under the assumption that the error function is quadratic. Line search techniques can be applied to approximate a minimization of the error function in the nonquadratic case, however line-search techniques are dependent on crucial user-defined parameters that determine when the line search should terminate [41].

One way to avoid a costly line search to determine the appropriate step size in determining an approximation of the minimum error function E is to estimate the term $\boldsymbol{s}_k = E''(\boldsymbol{w}_k)\boldsymbol{p}_k$ in the original conjugate gradient method with a non-symmetric approximation

$$\boldsymbol{s}_k = E''(\boldsymbol{w}_k)\boldsymbol{p}_k \approx \frac{E\prime(\boldsymbol{w}_k+\boldsymbol{\sigma}_k\boldsymbol{p}_k)-E\prime(\boldsymbol{w}_k)}{\boldsymbol{\sigma}_k}, \tag{63}$$

where $0 < \boldsymbol{\sigma}_k \ll 1$. The limit of the approximation tends to approach the true value of $E''(\boldsymbol{w}_k)\boldsymbol{p}_k$. The drawback of this modified conjugate gradient algorithm is known to converge to a non-stationary point, because the algorithm only works for error functions with positive definite Hessian matrices and the quadratic approximations which the algorithm uses can be very poor when the current point is too far from the desired minimum. In some cases, the Hessian matrix for the global error function E is indefinite in different areas of the weight space, causing the traditional conjugate gradient algorithm to fail for certain learning problems. One way to avoid this problem as proposed my Moller is to combine the model-trust region approach from the Levenberg-Marquardt algorithm [42], [43] with the conjugate gradient approach described earlier, resulting in the SCG algorithm. In the SCG case, a scalar λ_k is introduced, which helps regulate the indefiniteness of $E''(\boldsymbol{w_k})$, by setting

$$\boldsymbol{s}_k = \frac{E'(\boldsymbol{w}_k + \boldsymbol{\sigma}_k \boldsymbol{p}_k) - E'(\boldsymbol{w}_k)}{\boldsymbol{\sigma}_k} + \lambda_k \boldsymbol{p}_k \qquad (64)$$

and adjusting λ_k in each iteration in which $\boldsymbol{\delta}_k < 0$ indicates that the Hessian is not positive definite. The scalar λ_k can be adjusted to ensure that $\boldsymbol{\delta}_k > 0$, and thus ensuring that the Hessian $E''(\boldsymbol{w}_k)\boldsymbol{p}_k$ is positive definite. A reasonable choice of a raised λ_k, defined as $\overline{\lambda_k}$, in the quest for an optimal solution is

$$\lambda_k = 2\left(\lambda_k - \frac{\boldsymbol{\delta}_k}{|\boldsymbol{p}_k|^2}\right) \qquad (65)$$

leading to

$$\overline{\boldsymbol{\delta}}_k = -\boldsymbol{\delta}_k + \lambda_k |\boldsymbol{p}_k|^2 > 0 \qquad (66)$$

with the step size given by

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{\boldsymbol{p}_k^T + \lambda_k |\boldsymbol{p}_k|^2}. \qquad (67)$$

The values of the scalar λ_k are inversely proportional to the step size.

The output of the algorithm, which is the quadratic approximation to the error measure E_{qw} may not always be accurate since λ_k artificially scales the Hessian matrix, necessitating a mechanism to adjust the scalar to give a good approximation even if the

Hessian is positive definite. To implement this, a definition of a measure of how well $E_{qw}(\alpha_k \boldsymbol{p}_k)$ approximates $E'(\boldsymbol{w}_k + \alpha_k \boldsymbol{p}_k)$. One such measure is

$$\Delta_k = \frac{2\delta_k[E(\boldsymbol{w}_k) - E(\boldsymbol{w}_k + \alpha_k \boldsymbol{p}_k)]}{\mu_k^2} \tag{68}$$

with the condition that the closer Δ_k is to 1, the better the approximation, leading to the somewhat arbitrary decision boundary that

$$\text{if } \Delta_k > 0.75, \qquad \lambda_k = \frac{1}{4}\lambda_k \tag{69}$$

$$\text{if } \Delta_k < 0.25, \lambda_k = \lambda_k + \frac{\delta_k(1-\Delta_k)}{|\boldsymbol{p}_k|^2}. \tag{70}$$

The implementation details of the SCG algorithm are covered by Moller [41], and this algorithm is implemented by the MATLAB Neural Networks Toolbox, making it ideally suitable for MCU upset analysis.

Feedforward neural networks, such as the MLPs introduced earlier, are popular tools for nonlinear regression and classification problems. The feedforward neural network framework can be looked at from a Bayesian perspective, in which the choice of a neural network model can be viewed as defining a prior probability distribution over nonlinear functions. From the same perspective, the neural network learning process can be interpreted in terms of the posterior probability distribution over the unknown function. Additionally, in the limit of large but otherwise standard neural networks, the prior distribution over nonlinear functions implied by the Bayesian neural network falls in a class of probability distribution known as GPs [44]. Moreover, the hyperparameters of the neural network model determine the characteristic lengthscales of the GP. The concept of GPs for machine learning (GPML) is discussed next.

Chapter 3-4: Gaussian Processes for Machine Learning (GPML)

This section summarizes the theory behind GPML and summarizes approximation methods for GPML binary classification. The GP overview closely follows Rasmussen and Williams [45]. A variety of methods has been proposed to tackle the problem of supervised learning, two of which are more common than others. The first approach seeks to restrict the class of functions that are considered, whereas the second approach aims to give a prior probability to every possible function. Naturally, a higher probability is assigned to functions that are more likely because of some of their inherent characteristics, i.e. smoothness [45].

The issue with the first approach is that we are forced to decide upon the richness of features associated with a class of functions. If the choice of function does not represent the underlying data well, then it is obvious that the prediction capabilities of the learning algorithm will be poor. Increasing the flexibility of the class of functions being considered increases the risk of overfitting. The problem of overfitting occurs when the algorithm obtains a good fit for the training data but generalizes poorly to data that it has not seen before. The problem with the second approach is that many candidate functions are making it difficult to compute the results in a reasonable time. GPs effectively deal with both problems [45].

A GP is considered a generalization of the Gaussian probability distribution. A probability distribution describes random variables, whereas a stochastic process describes the properties of functions. A function can be considered as a very long vector, whose entries specify function values of $f(x)$ at a particular point. Considering the properties of the function of interest at a finite number of points, the inference step in GPs gives the same answer as if the entire set of infinitely many points were considered, making the problem computationally tractable. In general, many models used in machine learning and statistics can be considered to be special cases of GPs [45]. Effectively, GPs generalize multivariate Gaussian distributions over finite dimensional vectors to infinite dimensionality. Specifically, a GP is a stochastic process that has Gaussian-distributed finite-dimensional marginal distributions and it defines a distribution over functions, making each draw from a GP a function. GPs provide a principled, practical, and probabilistic approach to inference and learning in kernel machines [46].

A Gaussian probability distribution is completely specified by its mean and covariance matrix. Similarly, a GP is fully characterized by its mean function $m(x) = \boldsymbol{E}[f(x)]$ and covariance function

$$C(x, x') \coloneqq \boldsymbol{E}\big[\big(f(x) - m(x)\big)\big(f(x') - m(x')\big)\big]. \tag{71}$$

In general, a real process $f(x)$ is GP-distributed with a mean function $m(x)$ and a covariance function $C(x, x')$ is expressed as $f \sim \mathcal{GP}\big(m(x), C(x, x')\big)$ [46].

The mean function can be arbitrary for convenience, and it is often chosen to be a zero function because the observed outputs can always be centered to have a zero mean. The covariance function, on the other hand, must be positive-definite to ensure the existence of all finite-dimensional distributions. Positive-definiteness of the covariance function C ensures the positive semi-definiteness of all covariance matrices, Σ, in the exponent of the finite-dimensional multivariate Gaussian distribution [46].

Classification problems in machine learning deal with the case in which the requirement consists of assigning an input pattern $\boldsymbol{x}$ to one of $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_C\}$ classes. The classification case can be of binary type (two-class or $C = 2$) or multi-class ($C > 2$). The classification type provided by the GP method is probabilistic classification. In probabilistic classification, test prediction take the form of class probabilities, which contrasts with methods that provide a guess at the class label, such as SVMs [45].

Both classification and regression can be viewed in terms of function approximation problems. Solving classification problems using GPs is more demanding than solving regression problems using the same approach. The reason for this is that for regression problems, it was assumed that the likelihood function was Gaussian and a GP prior combined with a Gaussian likelihood results in a posterior GP over functions with everything remaining analytically tractable. For classification problems, in which class labels are discrete, the assumption of a Gaussian likelihood is not appropriate [45].

Linear models for binary classification are the foundation of GP classification models. In GP classification, output labels $y = 1$ (denoting upset) and $y = -1$ (denoting no upset) to distinguish the two classes in binary classification problems. The likelihood of upset is then given by

$$p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{x}^T \boldsymbol{w}) \tag{72}$$

given the weight vector $\boldsymbol{w}$ and $\sigma(z)$ can be any sigmoid. The probability of the two classes (upset/no-upset) sum to 1 and we have

$$p(y = 0|\boldsymbol{x}, \boldsymbol{w}) = 1 - p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = 1 - \sigma(\boldsymbol{x}^T \boldsymbol{w}). \tag{73}$$

In the case of symmetric likelihood functions, this can be written more compactly as

$$p(y_i|\boldsymbol{x_i}, \boldsymbol{w}) = \sigma(y_i f_i), \tag{74}$$

where $f_i \triangleq f(x_i) = \boldsymbol{x}_i^T \boldsymbol{w}$ [45].

Given a set of data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)|i = 1, \dots, n\}$ and assuming that the labels are generated independently, conditional on $f(x)$, a Gaussian prior $\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{0}, \Sigma_p\right)$, an un-normalized log-posterior can be obtained in the form of

$$\log p(\boldsymbol{w}|X, y) = -\frac{1}{2}\boldsymbol{w}^T \Sigma_p^{-1}\boldsymbol{w} + \sum_{i=1}^{n} \log \sigma(y_i f_i). \tag{75}$$

For the classification problem, the posterior does not have a simple analytic form. However, some sigmoid functions, i.e. the logistic and cumulative Gaussian the log likelihood is a concave function of $\boldsymbol{w}$ for fixed $\mathfrak{D}$, meaning that it is relatively straightforward to find its unique maximum. The standard method for finding the maximum is Newton's method also called the iteratively reweighted least squares (IRLS), however faster methods may exist such as the conjugate gradient ascent [45].

In the case of binary classification, i.e. upset/no-upset, the basic idea behind the GP approach is simple. A GP prior is placed over the latent function $f(x)$ and this is pushed through the logistic function to obtain a prior on $\pi(x) \triangleq p(y = 1|x) = \sigma\left(f(x)\right)$. Model development for binary classification is a natural generalization of the linear logistic model for regression. Specifically, the linear $f(x)$ function in this equation

$$\log p(\boldsymbol{w}|X,y) = -\frac{1}{2}\boldsymbol{w}^T\Sigma_p^{-1}\boldsymbol{w} + \sum_{i=1}^{n}\log\sigma(y_i f_i) \tag{76}$$

is replaced by a GP and correspondingly the Gaussian prior on the weights by a GP prior [45].

More specifically, to format the implementation of GP for binary classification (GPC), we let the output labels $y = \{-1,1\}$ corresponding to an input vector $\boldsymbol{x}$**.** In this case, the GPC model is discriminative in the sense that it models $p(y|\boldsymbol{x})$ which for fixed x is a Bernoulli distribution. The probability of MCU upset (success) is $p(y = 1|\boldsymbol{x})$ is directly related to an unconstrained latent function $f(x)$ which is mapped to the unit interval by a sigmoidal transformation, i.e. the $logit$ or the $probit$ functions. For analytic convenience in the EP algorithm, the $probit$ model is used in the form of $p(y = 1|\boldsymbol{x}) = \Phi\big(f(\boldsymbol{x})\big)$ where Φ denotes the cumulative density function (CDF) of the standard normal distribution [47].

In the GPC model, Bayesian inference is performed on the latent function f given the observed data $\mathfrak{D} = \{(y_i, \boldsymbol{x}_i)|i = 1, \dots, m\}$. Let $f_i = f(\boldsymbol{x}_i)\ and\ \boldsymbol{f} = [f_1, \dots, f_m]^T$ represent the values of the latent function and $\boldsymbol{y} = [y_1, \dots, y_m]^T$**,** $\boldsymbol{x} = [x_1, \dots, x_m]^T$ represent the class labels (target outputs) and inputs, respectively. Given the latent functions, the class labels are interdependent Bernoulli variables, so the joint likelihood is

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{i=1}^{m} p(y_i|f_i) \tag{77}$$

and it depends on f only through its value at the corresponding observed inputs. In the case of the $probit$ model, the individual likelihood terms become $p(y_i|f_i) = \Phi(y_i|f_i)$ [47].

As the prior over functions f a zero-mean GP prior is used, a GP is a stochastic process where each input $\boldsymbol{x}$ has an associated random variable $f(\boldsymbol{x})$. The joint distribution of function values that corresponds to any set of inputs $\boldsymbol{X}$ is multivariate Gaussian as in $p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})$. The covariance matrix is defined element-wise so that $\boldsymbol{K}_{ij} = k(x_i, x_j, \Theta)$ where k is a positive-definite covariance function parameterized by Θ. By choosing a covariance function we inherently introduce hyperparameters Θ to the prior [47].

Next, using Bayes' rule, the posterior distribution over latent function values $\boldsymbol{f}$ for given hyperparameters θ becomes

$$p(\boldsymbol{f}|\mathcal{D}, \theta) = \frac{p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta})}{p(\mathcal{D}|\boldsymbol{\theta})} = \frac{\mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})}{p(\mathcal{D}|\boldsymbol{\theta})} \prod_{i=1}^{m} \Phi(y_i, f_i), \tag{78}$$

which is non-Gaussian [47].

The main purpose of classification models is to predict the class label y_* for test inputs $\boldsymbol{x}_*$**.** The distribution of the latent function value can be computed by marginalization

$$p(f_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*) = \int p(f_*|\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{x}_*)p(\boldsymbol{f}|\mathcal{D}, \boldsymbol{\theta})d\boldsymbol{f}\,; \tag{79}$$

moreover, by computing the expectation

$$p(y_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*) = \int p(y_*|f_*)p(f_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*)df_* \tag{80}$$

by which the prediction distribution is obtained, which is a Bernoulli distribution [47].

According to Kuss et al. [47], a key premise is that a Gaussian approximation to the posterior implies a GP approximation to the posterior process, resulting in an approximate predictive distribution for test cases. Introducing the approximate Gaussian posterior $p(f_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*)$ above, gives the approximate posterior

$$q(f_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*) = \mathcal{N}(f_*|\mu_*) \tag{81}$$

with mean and variance defined as

$$\mu_* = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{m} \tag{82}$$

$$\sigma_*^2 = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T (\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1} \boldsymbol{A} \boldsymbol{K}^{-1}) \boldsymbol{k}_*, \tag{83}$$

where $\boldsymbol{k}_* = [k(\boldsymbol{x}_1, \boldsymbol{x}_*), \dots, k(\boldsymbol{x}_m, \boldsymbol{x}_*)]^T$ is a vector of prior covarianes between $\boldsymbol{x}_*$ and the training inputs $\boldsymbol{X}$. For the $probit$ likelihood the approximate predictive probability $p(y_*|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*)$ of $\boldsymbol{x}_*$ belonging to class 1 can be analytically computed as

$$q(y_* = 1|\mathcal{D}, \boldsymbol{\theta}, \boldsymbol{x}_*) = \int \Phi(f_*) \mathcal{N}(f_*|\mu_*, \sigma_*^2) = \Phi\left(\frac{\mu_*}{\sqrt{1+\sigma_*^2}}\right). \tag{84}$$

The parameters $\boldsymbol{m}$ and $\boldsymbol{A}$ of the posterior approximation can be found using Laplace's method or by Expectation Propagation (EP), introduced in the next two subsections respectively.

Chapter 3-4-1: Laplace's Method

Since the posterior, the predictive distribution, nor the marginal likelihood, can be computed analytically, approximations are needed. This subsection covers the theory behind Laplace's approximation method. Laplace's method uses a Gaussian approximation $q(\boldsymbol{f}|X, \boldsymbol{y})$ of the posterior $p(\boldsymbol{f}|X, \boldsymbol{y})$ in $p(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*) =$ $\int p(f_*|\boldsymbol{f}, \boldsymbol{X}, \boldsymbol{x}_*, \boldsymbol{f})p(\boldsymbol{f}|X, \boldsymbol{y})d\boldsymbol{f}$, obtained by

$$q(\boldsymbol{f}|X, \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{f}|\hat{\boldsymbol{f}}, A^{-1}\right) \propto exp\left(-\frac{1}{2}\left(\boldsymbol{f} - \hat{\boldsymbol{f}}\right)^T A\left(\left(\boldsymbol{f} - \hat{\boldsymbol{f}}\right)\right)\right), \qquad (85)$$

where $\hat{\boldsymbol{f}} = \text{argmax}_{\boldsymbol{f}}\, p(\boldsymbol{f}|X, \boldsymbol{y})$ and $A = -\nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y})_{\boldsymbol{f}=\hat{\boldsymbol{f}}}$ is the Hessian of the negative log posterior at that point. What is needed next is to determine $\hat{\boldsymbol{f}}$ and A, then to make predictions after obtaining $q(\boldsymbol{f}|\boldsymbol{y})$ [45].

Bayes' rule gives the posterior over the latent variables by computing

$$p(\boldsymbol{f}|X, \boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|X)}{p(\boldsymbol{y}|X)}; \qquad (86)$$

however, with $p(\boldsymbol{y}|X)$ being independent of $\boldsymbol{f}$, only the un-normalized posterior will be considered. Taking the logarithm of the expression above and introducing the expression $\log\left(p(\boldsymbol{f}|X)\right) = -\frac{1}{2}\boldsymbol{f}^T K^{-1} f - \frac{1}{2}\log|K| - \frac{n}{2}\log(2\pi)$ for the GP prior,

$$\psi(\boldsymbol{f}) \triangleq \log p(\boldsymbol{y}|\boldsymbol{f}) + \log p(\boldsymbol{f}|X)$$

$$= \log p(\boldsymbol{y}|\boldsymbol{f}) - \frac{1}{2}\boldsymbol{f}^T K^{-1} f - \frac{1}{2}\log|K| - \frac{n}{2}\log(2\pi). \qquad (87)$$

Differentiating the expression above and determining the maximum, we obtain $\hat{\boldsymbol{f}} =$ $K\left(\nabla \log p\left(\boldsymbol{y}|\hat{\boldsymbol{f}}\right)\right)$, which needs to be solved iteratively via methods such as the Newton's method. Having determined the maximum posterior $\hat{\boldsymbol{f}}$**,** the Laplace approximation to the posterior is given as a Gaussian with mean $\hat{\boldsymbol{f}}$ covariance matrix given by the negative inverse Hessian of ψ

$$q(\boldsymbol{f}|X, \boldsymbol{y}) = \mathcal{N}\big(\hat{\boldsymbol{f}}, (K^{-1} + W)^{-1}\big), \tag{88}$$

where $\boldsymbol{A} = (K^{-1} + W)^{-\mathbf{1}}$ [45].

After calculating the posterior, the next step in using Laplace's method is to determine the computation of the predictive step. Under the Laplace approximation, the posterior mean for f_* can be expressed as

$$\mathbb{E}_q[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = \boldsymbol{k}(\boldsymbol{x}_*)^T K^{-1}\hat{\boldsymbol{f}} = \boldsymbol{k}(\boldsymbol{x}_*)^T \nabla \log p\big(\boldsymbol{y}|\hat{\boldsymbol{f}}\big). \tag{89}$$

In this case, positive training examples result in a positive coefficient for their kernel function, because $\nabla_i \log p(y_i|f_i) > 0$, while negative examples result in negative coefficients, similar to solutions obtained via SVMs. It is also worth noting that training examples, for which $\nabla_i \log p(y_i|f_i) \cong 0$, do not contribute strongly to predictions at novel test points. It is also possible to compute the variance of $f_*|X, \boldsymbol{y}$ under the Gaussian approximation, e.g.

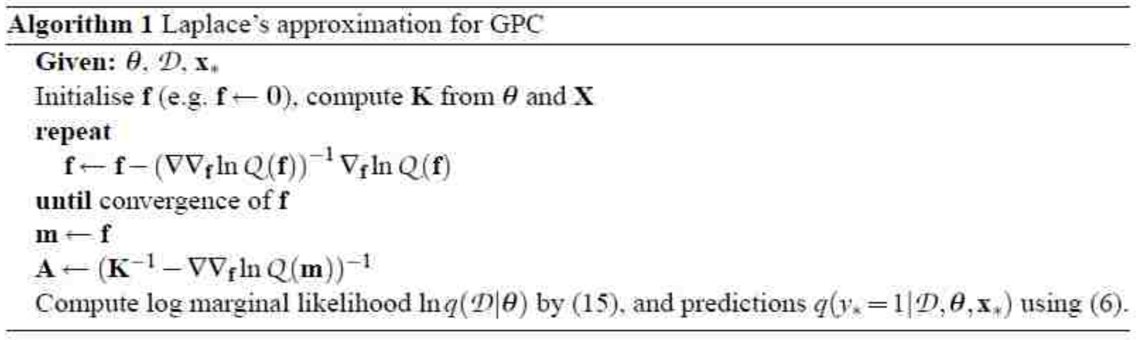
$$\mathbb{V}_q[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T(K + W^{-1})^{-1}\boldsymbol{k}_*. \tag{90}$$

Having obtained the mean and variance of f_*, predictions are computed by

$$\bar{\pi}_* \cong \mathbb{E}_q[\pi_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = \int \sigma(f_*) q(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*) df_* \tag{91}$$

in which $q(f_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$ is Gaussian with mean and variance given by $\mathbb{E}_q[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*]$ and $\mathbb{V}_q[f_*|X, \boldsymbol{y}, \boldsymbol{x}_*]$, respectively. In this case, if $\sigma(z)$ is the cumulative Gaussian function then this prediction can be computed analytically; however, if it is anything else but

Gaussian, we need to resort to sampling or approximation methods to compute the one-dimensional integral. If we are only concerned about the most probable classification, it is not necessary to compute the prediction step $\mathbb{E}_q[\pi_*|X, \boldsymbol{y}, \boldsymbol{x}_*]$; however, the step is required if we are interested in confidence intervals related to our prediction. The algorithm for Laplace's approximation for GP classification is shown in Fig. 42 [45].



Algorithm 1 Laplace's approximation for GPC

```
Given: $\theta, \mathcal{D}, \mathbf{x}_*$
Initialise $\mathbf{f}$ (e.g. $\mathbf{f} \leftarrow 0$), compute $\mathbf{K}$ from $\theta$ and $\mathbf{X}$
repeat
    $\mathbf{f} \leftarrow \mathbf{f} - (\nabla\nabla_{\mathbf{f}} \ln Q(\mathbf{f}))^{-1} \nabla_{\mathbf{f}} \ln Q(\mathbf{f})$
until convergence of $\mathbf{f}$
$\mathbf{m} \leftarrow \mathbf{f}$
$\mathbf{A} \leftarrow (\mathbf{K}^{-1} - \nabla\nabla_{\mathbf{f}} \ln Q(\mathbf{m}))^{-1}$
Compute log marginal likelihood $\ln q(\mathcal{D}|\theta)$ by (15), and predictions $q(y_* = 1|\mathcal{D}, \boldsymbol{\theta}, \mathbf{x}_*)$ using (6).
```

Figure 42: Laplace's method for Gaussian process classification [47].

A problem with Laplace's approximation is that it is uncontrolled, meaning that the Hessian evaluated at $\hat{\boldsymbol{f}}$ may give a poor approximation to the true shape of the posterior. An alternative to Laplace's approximation for GPC is expectation propagation (EP), which is discussed next.

Chapter 3-4-2: Expectation Propagation

The EP algorithm, coined by Minka in 2001 [48], is a general approximation tool that can be used in a wide range of applications, including the specific case of a GP model for binary classification. EP is used in Bayesian machine learning to tune the parameters of a simpler, albeit approximate distribution, e.g. a Gaussian, to match the posterior distribution of the true model parameters given the experimental data. It is a deterministic method well-suited for large datasets as well as dynamic systems, where exact Bayesian inference methods fail and where Monte Carlo methods are too slow [46].

The posterior distribution over latent variables

$$p(f|X,\boldsymbol{y}) = \frac{1}{Z} p(\boldsymbol{f}|X) \prod_{i=1}^{n} p(y_i|f_i), \tag{92}$$

is a concept of high importance in this approach, in which the posterior is computed using Bayes' rule as the product of a normalization term, $\frac{1}{Z}$, the prior, $p(\boldsymbol{f}|X)$, and the likelihood $\prod_{i=1}^{n} p(y_i|f_i)$. The prior is assumed to be Gaussian and the likelihood factorizes over training cases. The normalization term is the marginal likelihood given by

$$Z = p(\boldsymbol{y}|X) = \int p(f|X) \prod_{i=1}^{n} p(y_i|f_i) d\boldsymbol{f}. \tag{93}$$

The difference between the regression and classification analytic framework for GPs is that in the case of classification, the likelihood $p(y_i|f_i)$ is not Gaussian – a property that was heavily relied on in arriving at analytical solution for the GP regression framework. When the probit likelihood, $p(y_i|f_i) = \Phi(y_i|f_i)$, is used for binary classification, it makes the posterior $p(f|X,\boldsymbol{y})$ above, intractable. To overcome this intractability in the EP framework, the likelihood is approximated by a local likelihood approximation in the form of an un-normalized Gaussian function in f_i,

$$p(y_i|f_i) \cong t_i\left(f_i|\tilde{Z}_i, \tilde{\mu}_i, \tilde{\sigma}_i^2\right) \triangleq \tilde{Z}_i \mathcal{N}(f_i|\tilde{\mu}_i, \tilde{\sigma}_i^2) \tag{94}$$

where $\tilde{Z}_i, \tilde{\mu}_i$, and $\tilde{\sigma}_i^2$ are site parameters, and $\mathcal{N}$ is the normalized Gaussian distribution [45]. It is clear that we are approximating a likelihood or a probability distribution which normalizes over the targets y_i, by an unnormalized Gaussian distribution over the latent variables f_i.

The EP method unifies two previously developed techniques: assumed density filtering (ADF), which extends the Kalman filter, and loopy belief propagation, which extends belief propagation in Bayesian networks. EP approximates the belief states by only retaining expectation, such as the mean and variance, and it keeps iterating until these expectations are consistent throughout the network [48].

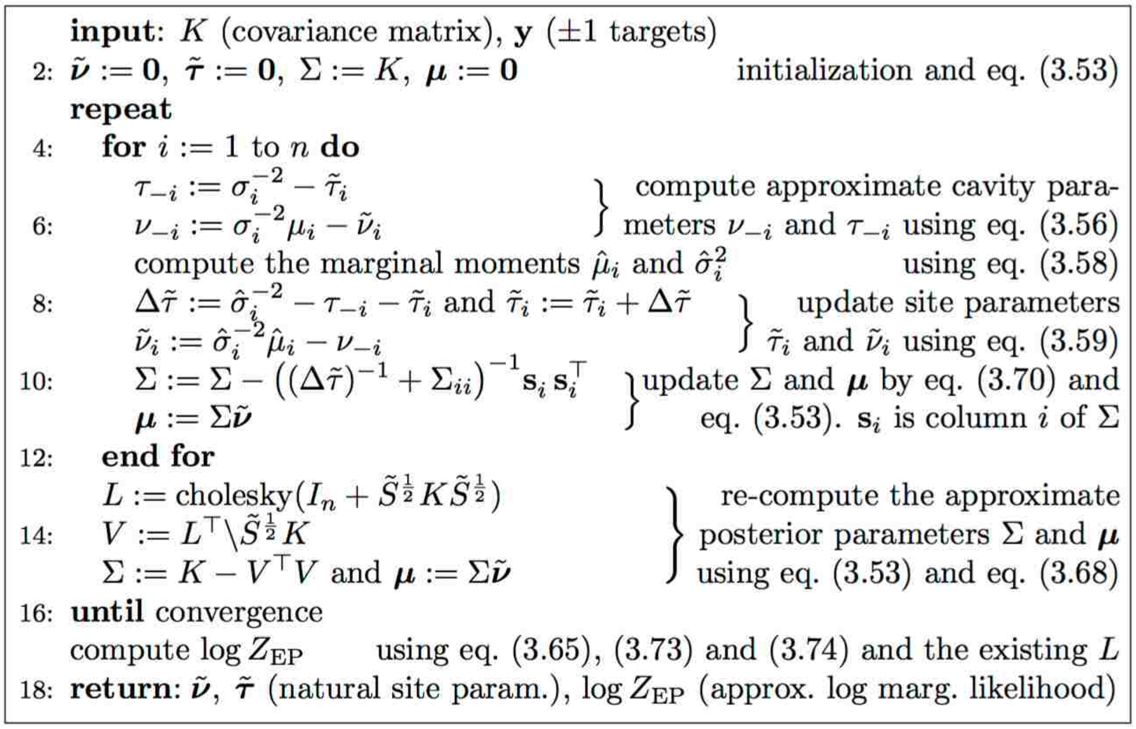
EP is an extension of ADF, which is a single-pass, sequential method for computing an approximate posterior distribution. In ADF, observations are processed one-by-one, updating the posterior distribution which is then approximated before processing the next observation. The primary weakness of ADF is its sequential nature of processing – information that is discarded early may prove to be important later during processing. An additional weakness is that ADF may be sensitive to observation ordering, which is an undesirable artifact in a batch-processing context [48].

Specifically, EP extends ADF to incorporate iterative refinement of the approximations by making additional passes through the network. Information from later observation refines the choices made earlier so that the most relevant information is retained. EP is faster than sampling and more general than Kalman filtering, and it is applicable in all scenarios in which ADF applies as well. Computationally, it is more expensive than ADF but only by a constant factor that is proportional to the number of passes EP performs in the network [48].

In loopy belief networks, it is known that approximate marginal distributions can be obtained by iterating belief propagation recursions – known as loopy belief propagation. EP is more general than loopy belief propagation in two ways: (1) like variational methods, it can use approximations that are not completely disconnected, and it can impose useful constraints on functional form, such as a multivariate Gaussian. Minka [48] describes the general form of EP as follows:

1. Initialize the term approximations $\bar{t}_i$
2. Compute the posterior for $\boldsymbol{x}$ from the product of $\bar{t}_i$
   $$q(\boldsymbol{x}) = \frac{\prod_i \bar{t}_i(x)}{\int \prod_i \bar{t}_i(x) dx}$$
3. Until all $\bar{t}_i$ converge:
   a. Choose a $\bar{t}_i$ to refine
   b. Remove $\bar{t}_i$ from the posterior to get an 'old' posterior $q^{\backslash i}(\boldsymbol{x})$ by dividing and normalizing:
   $$q^{\backslash i}(\boldsymbol{x}) \propto \frac{q(\boldsymbol{x})}{\bar{t}_i(\boldsymbol{x})}$$
   c. Combine $q^{\backslash i}(\boldsymbol{x})$ and $\bar{t}_i(\boldsymbol{x})$ and minimize KL-divergence to get a new posterior $q(\boldsymbol{x})$ with normalizer Z_i
   d. Update $\bar{t}_i = Z_i q(\boldsymbol{x}) / q^{\backslash i}(\boldsymbol{x})$
4. Use the normalizing constant of $q(\boldsymbol{x})$ as an approximation to $p(D)$:
   $$p(D) \approx \int \prod_i \bar{t}_i(\boldsymbol{x}) d\boldsymbol{x}$$

which for the binary classification case, is implemented as described in Fig. 43.

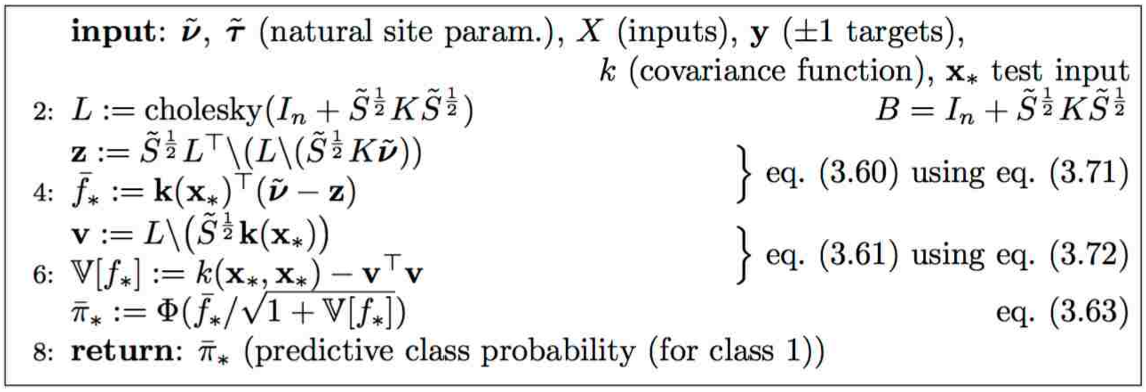


```
    input: $K$ (covariance matrix), $\mathbf{y}$ ($\pm 1$ targets)
2:  $\tilde{\boldsymbol{\nu}} := \mathbf{0},\ \tilde{\boldsymbol{\tau}} := \mathbf{0},\ \Sigma := K,\ \boldsymbol{\mu} := \mathbf{0}$    initialization and eq. (3.53)
    repeat
4:      for $i := 1$ to $n$ do
            $\tau_{-i} := \sigma_i^{-2} - \tilde{\tau}_i$    } compute approximate cavity para-
6:          $\nu_{-i} := \sigma_i^{-2}\mu_i - \tilde{\nu}_i$    } meters $\nu_{-i}$ and $\tau_{-i}$ using eq. (3.56)
            compute the marginal moments $\hat{\mu}_i$ and $\hat{\sigma}_i^2$    using eq. (3.58)
8:          $\Delta\tilde{\tau} := \hat{\sigma}_i^{-2} - \tau_{-i} - \tilde{\tau}_i$ and $\tilde{\tau}_i := \tilde{\tau}_i + \Delta\tilde{\tau}$    } update site parameters
            $\tilde{\nu}_i := \hat{\sigma}_i^{-2}\hat{\mu}_i - \nu_{-i}$    } $\tilde{\tau}_i$ and $\tilde{\nu}_i$ using eq. (3.59)
10:         $\Sigma := \Sigma - \big((\Delta\tilde{\tau})^{-1} + \Sigma_{ii}\big)^{-1}\mathbf{s}_i\mathbf{s}_i^\top$    } update $\Sigma$ and $\boldsymbol{\mu}$ by eq. (3.70) and
            $\boldsymbol{\mu} := \Sigma\tilde{\boldsymbol{\nu}}$    } eq. (3.53). $\mathbf{s}_i$ is column $i$ of $\Sigma$
12:     end for
        $L := \text{cholesky}(I_n + \tilde{S}^{\frac{1}{2}}K\tilde{S}^{\frac{1}{2}})$    } re-compute the approximate
14:     $V := L^\top \backslash \tilde{S}^{\frac{1}{2}}K$    } posterior parameters $\Sigma$ and $\boldsymbol{\mu}$
        $\Sigma := K - V^\top V$ and $\boldsymbol{\mu} := \Sigma\tilde{\boldsymbol{\nu}}$    } using eq. (3.53) and eq. (3.68)
16: until convergence
    compute $\log Z_{\text{EP}}$    using eq. (3.65), (3.73) and (3.74) and the existing $L$
18: return: $\tilde{\boldsymbol{\nu}}$, $\tilde{\boldsymbol{\tau}}$ (natural site param.), $\log Z_{\text{EP}}$ (approx. log marg. likelihood)
```

Figure 43: Expectation propagation for binary classification [45].

The classification targets $\boldsymbol{y}$ are used in line 7. In lines 13-15, the algorithm re-computes the approximate posterior. The re-computation is performed again because of the large number of rank-one updates in line 10 causing a loss in numerical precision in Σ. The computational complexity of the EP algorithm is dominated but the rank one-updates in line 10 and is in the order of $\mathcal{O}(n^2)$ per variable or $\mathcal{O}(n^3)$ when sweeping over all variables, indicating that large datasets will take considerably longer to process, making this a potentially expensive process in terms of computational time [45].

For the second (prediction step) the algorithm is described in Fig. 44.



```
   input: $\tilde{\boldsymbol{\nu}}$, $\tilde{\boldsymbol{\tau}}$ (natural site param.), $X$ (inputs), $\mathbf{y}$ ($\pm 1$ targets),
                                   $k$ (covariance function), $\mathbf{x}_*$ test input
2: $L := \text{cholesky}(I_n + \tilde{S}^{\frac{1}{2}} K \tilde{S}^{\frac{1}{2}})$          $B = I_n + \tilde{S}^{\frac{1}{2}} K \tilde{S}^{\frac{1}{2}}$
   $\mathbf{z} := \tilde{S}^{\frac{1}{2}} L^\top \backslash (L \backslash (\tilde{S}^{\frac{1}{2}} K \tilde{\boldsymbol{\nu}}))$          } eq. (3.60) using eq. (3.71)
4: $\bar{f}_* := \mathbf{k}(\mathbf{x}_*)^\top (\tilde{\boldsymbol{\nu}} - \mathbf{z})$
   $\mathbf{v} := L \backslash (\tilde{S}^{\frac{1}{2}} \mathbf{k}(\mathbf{x}_*))$          } eq. (3.61) using eq. (3.72)
6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
   $\bar{\pi}_* := \Phi(\bar{f}_* / \sqrt{1 + \mathbb{V}[f_*]})$          eq. (3.63)
8: return: $\bar{\pi}_*$ (predictive class probability (for class 1))
```

Figure 44: Algorithm for prediction step of EP [45].

As seen in line 8, the prediction algorithm returns

$$\overline{\pi}_* \triangleq p(y_* = 1 | X, \boldsymbol{y}, \boldsymbol{x}_*) = \int \sigma(f_*) p(f_* | X, \boldsymbol{y}, \boldsymbol{x}_*) df_*, \qquad (95)$$

which represents the predictive class probability for class 1, representing MCU upset probability. There is no guarantee for the algorithm to converge, but several authors have reported that EP for GP models works relatively well [45].

The assumption that the posterior is close to a Gaussian as it is usually the case with low-dimensional parametric models with large amounts of data is not likely to be valid with a high-dimensional model with relatively few data. Laplace's approximation technique centers around the mode of the posterior resulting in overly cautious predictive distributions, whereas EP does not rely on a local expansion. EP assumes the marginal distribution of the posterior is well approximated by Gaussians and Kuss et al. [47] show that it performs better than Laplace's approximation for a generic set of pseudo-randomly generated data.

Since the mean and covariance functions are key in fully specifying a GP and having discussed the theory behind GP classification, the next two subsections cover the background of the covariance function and model selection for GPML.

Chapter 3-4-3: Covariance Functions for GPML

The covariance function is the critical ingredient in the GP predictor because it encodes some assumptions about the function which we want to infer. Generally, in a supervised learning problem, the concept of similarity between data points is key with the underlying assumption that data points with input $\boldsymbol{x}$ which are close are likely to have similar outcomes or target values y and therefore the data points used for testing the model that are near a training point should be able to predict the outcome of the test point with some probability. In GPML, the covariance function defines this similarity between data points. An arbitrary function of input pairs $\boldsymbol{x}$ and $\boldsymbol{x}'$ is generally not a valid covariance function, and this subsection describes the general properties of valid covariance functions. Supervised machine learning problems by nature share some general mathematical concepts, therefore each section may repeat some concepts covered earlier, however will be described as they apply to a machine learning discipline [45].

A stationary covariance function $\boldsymbol{x} - \boldsymbol{x}'$ is invariant if it is not sensitive to translations in the input space. For example, the squared exponential (SE) function, that specifies the covariance between pairs of random variables,

$$\operatorname{cov}\left(f(\boldsymbol{x}_p), f(\boldsymbol{x}_q)\right) = k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \exp(-\frac{1}{2}|\boldsymbol{x}_p - \boldsymbol{x}_q|^2) \qquad (96)$$

is a stationary covariance function, because it is a function of $\boldsymbol{x}_p - \boldsymbol{x}_q$. Additionally, if the covariance function is also a function $|\boldsymbol{x} - \boldsymbol{x}'|$ it is also isotropic, meaning that it is invariant to all rigid motions. The SE function is an example of an isotropic function. Furthermore, the function is considered a radial basis function (RBF) if it is a function of $r = |\boldsymbol{x} - \boldsymbol{x}'|$ [45].

In the case where a covariance function depends only on $\boldsymbol{x}$ and $\boldsymbol{x}'$ through $\langle \boldsymbol{x} \cdot \boldsymbol{x}' \rangle$**,** it is labeled a dot product covariance function, an example of which is $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_0^2 + \langle \boldsymbol{x} \cdot \boldsymbol{x}' \rangle$**.** The dot product covariance function is obtained through linear regression by putting $\mathcal{N}(0,1)$ priors on the coefficients of $x_d(d = 1, \ldots, D)$ and a prior of $\mathcal{N}(0, \sigma_0^2)$ on the bias or the constant function. An important example of a dot product covariance function is the inhomogeneous polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_0^2 + \langle \boldsymbol{x} \cdot \boldsymbol{x}' \rangle$, where p is a positive integer. The dot product covariance functions are invariant to a rotation of the coordinates about the origin, but are not invariant to translations [45].

The general name for a function $k(\cdot)$ of two arguments that maps a pair of inputs $\boldsymbol{x} \in X, \boldsymbol{x}' \in X$ into $\mathbb{R}$ is a kernel. The term stems from integral operator theory, where the operator

$$T_k f(x) = \int_X k(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}') d\mu\,(\boldsymbol{x}') \tag{97}$$

and μ denotes a measure. A kernel is said to be symmetric if $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$, representing a key criterion for a valid covariance function.

Just as in SVMs, the Gram matrix is a fundamental concept in GPML. If we consider a set of input points $\{x_i | i = 1, \ldots, n\}$, the Gram matrix contains entries determined by $K_{ij} = k(x_i, x_j)$. In the case where k is a covariance function, the corresponding matrix K is referred to as the covariance matrix. A real $n \times n$ matrix K is called positive semidefinite (PSD) if it satisfies $Q(\boldsymbol{v}) = \boldsymbol{v}^T K \boldsymbol{v} \geq 0$ for all $v \in \mathbb{R}^n$, where $Q(\boldsymbol{v})$ is referred to as the quadratic form. Generally, the Gram matrix belonging to a general kernel function does not need to be PSD, but the Gram matrix belonging to a covariance function in GPML must be PSD. Also, a kernel k is said to be PSD if

$$\int k(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}) f(\boldsymbol{x}') d\mu(\boldsymbol{x}) d\mu(\boldsymbol{x}') \geq 0 \tag{98}$$

for all $f \in L_2(X, \mu)$ and every time a kernel function gives rise to PSD Gram matrices for any $n \in (\mathbb{N}, \mathcal{D})$, that kernel function is considered to be PSD [45].

Another key concept in theory related to valid covariance functions for GPML is that of mean square continuity, or smoothness, and differentiability. If we consider $\boldsymbol{x}_1, \boldsymbol{x}_2, ...$ to be a sequence of points and $\boldsymbol{x}_*$is a fixed point in $\mathbb{R}^{\mathcal{D}}$ such that $|\boldsymbol{x}_k - \boldsymbol{x}_*| \to 0$ as $k \to \infty$, then process $f(\boldsymbol{x})$ is continuous in mean square at $\boldsymbol{x}_*$ if $\mathbb{E}[|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_*)|^2] \to 0$ as $k \to \infty$. If both conditions are met for all $\boldsymbol{x}_* \in A$ with $A \subset \mathbb{R}^{\mathcal{D}}$ then $f(\boldsymbol{x})$ is said to be continuous in mean square at $\boldsymbol{x}_*$, which for stationary covariance functions reduces to checking continuity at $k(\boldsymbol{0})$ [45].

The mean square derivative of $f(\boldsymbol{x})$ in the ith direction is defined as

$$\frac{df(\boldsymbol{x})}{dx_i} = \underset{h \to 0}{\mathrm{l.i.m}} \frac{f(\boldsymbol{x} + h\boldsymbol{e}_i)}{h},$$

when the limit exists and where l.i.m denotes the limit in mean square and $\boldsymbol{e}_i$ is the unit vector in the ith direction. Here, the covariance function of $df(\boldsymbol{x})/dx_i$ is given by $\partial^2 k(\boldsymbol{x}, \boldsymbol{x}')/\partial x_i \partial x'_i$ and these definitions can be extended to higher order derivatives [45].

The input domain for covariance functions is $X \subset \mathbb{R}^{\mathcal{D}}$. A commonly used covariance function is the squared exponential (SE) covariance function of the form

$$k_{SE}(r) = \exp\left(-\frac{r^2}{2l^2}\right), \tag{99}$$

where the parameter l defines the characteristic length scale. The SE covariance function is infinitely differentiable, meaning that it is very smooth. Even though there are arguments that such strong smoothness is not realistic for modeling real world physical processes along with a recommendation to use the Matérn class [49], the SE is one of the most widely used kernels within the kernel machine learning field [45].

The Matérn class of covariance functions has the form of

$$k_{\text{Matérn}}(r) = \frac{2^{l-v}}{\Gamma(v)}\left(\frac{\sqrt{2v}r}{l}\right)^{v} K_v\left(\frac{\sqrt{2v}r}{l}\right) \tag{100}$$

where v and l denote positive parameters, and K_v is a modified Bessel function. If the scaling for the Matérn class is chosen in such a way that $v \to \infty$, the result is the SE covariance function. In the Matérn class, the process is k-times MS differentiable if and only if $v > k$. Also, if $v = p + 1/2$, with p being a nonnegative integer, the Matérn covariance function transforms into a product of an exponential covariance function and a polynomial covariance function of order p. Rasmussen and Williams [45] postulate that the most interesting cases of the Matérn class for machine learning may be with $v = 3/2$ and $v = 5/2$, for which

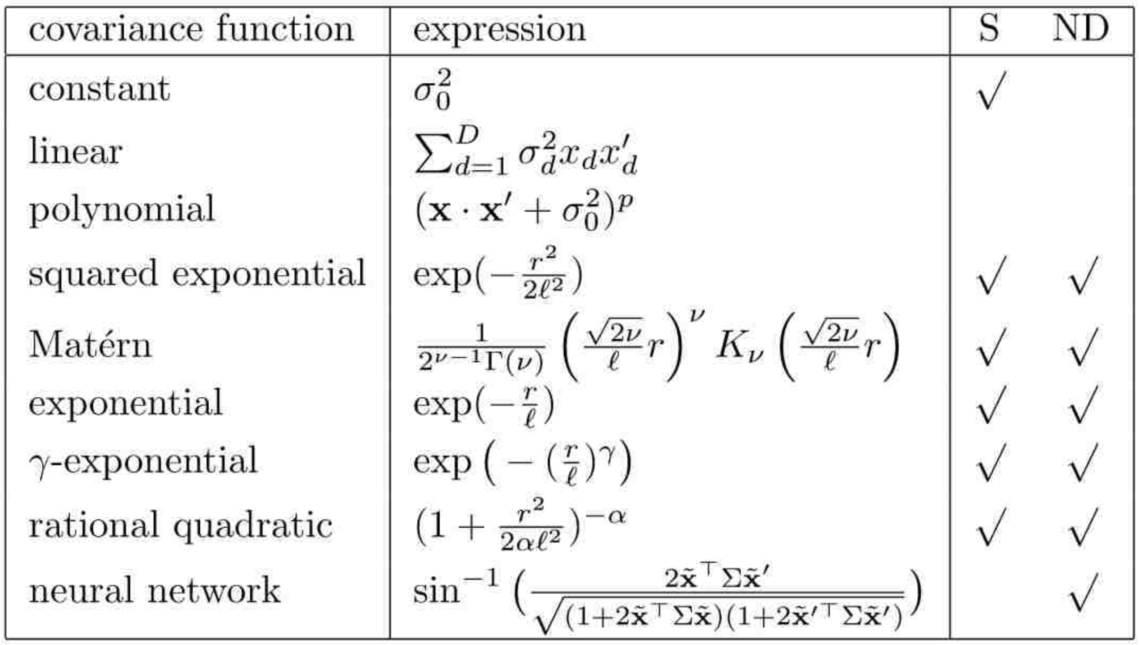
$$k_{v=3/2}(r) = \left(1 + \frac{\sqrt{3}r}{l}\right) exp\left(-\frac{\sqrt{3}r}{l}\right), \tag{101}$$

$$k_{v=5/2}(r) = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) exp\left(-\frac{\sqrt{5}r}{l}\right). \tag{102}$$

The reason for the machine learning suitability at these two values is that for $v = 1/2$, the process becomes very rough and for $v \geq 7/2$ it becomes difficult to distinguish between finite values of v in the presence of noise. Additionally, as mentioned earlier, as $v \to \infty$, the Matérn class becomes the SE covariance function.

Table 8 summarizes several commonly used covariance functions. The table shows the covariance functions expressed in terms of $\boldsymbol{x}$ and $\boldsymbol{x}'$ or as a function of $\boldsymbol{r} = |\boldsymbol{x} - \boldsymbol{x}'|$. The columns denoted S and ND denote whether the covariance functions are stationary and nondegenerate, respectively.

Table 8: Commonly used covariance functions [45].



covariance function	expression	S	ND
constant	σ_0^2	✓	
linear	$\sum_{d=1}^{D} \sigma_d^2 x_d x'_d$		
polynomial	$(\mathbf{x} \cdot \mathbf{x}' + \sigma_0^2)^p$		
squared exponential	$\exp(-\frac{r^2}{2\ell^2})$	✓	✓
Matérn	$\frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} r\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}}{\ell} r\right)$	✓	✓
exponential	$\exp(-\frac{r}{\ell})$	✓	✓
γ-exponential	$\exp\left(-(\frac{r}{\ell})^{\gamma}\right)$	✓	✓
rational quadratic	$(1 + \frac{r^2}{2\alpha\ell^2})^{-\alpha}$	✓	✓
neural network	$\sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^{\top}\Sigma\tilde{\mathbf{x}}'}{\sqrt{(1+2\tilde{\mathbf{x}}^{\top}\Sigma\tilde{\mathbf{x}})(1+2\tilde{\mathbf{x}}'^{\top}\Sigma\tilde{\mathbf{x}}')}}\right)$		✓

Chapter 3-4-4: Model Selection for GP Classification

Many practical applications do not lend themselves to an easy specification of all aspects of covariance functions with confidence. Some properties, such as the stationarity of the covariance function may be easily determined from the context, other properties such as the value of hyperparameters, e.g. length-scales, are more difficult. Moreover, the exact form and possible hyperparameters of the likelihood function may not be known in advance. Thus, to use GPs as a potential analysis tool in the MCU upset problem, it is necessary to address model selection techniques. The model selection problem may be interpreted in a broad sense to include the discrete choice of the covariance function form in addition to the values for any hyperparameters.

For the model to become a practical tool in an application, one is required to make choices regarding some of the details of its specification. While it may be easy to specify some properties of the model, only vague information may be available about some other aspects. As mentioned earlier, model selection refers to the discrete choices and the setting of hyperparameters of the covariance functions. A good choice for a model can help to refine its predictions and give the user a valuable interpretation of the experimental data properties, e.g. that a non-stationary covariance function may perform better than a stationary one.

In the previous subsection, an overview of available covariance functions was presented. Each of these covariance functions is associated with several free hyperparameters whose values need to be determined. Thus, selecting a covariance function for a particular application comprises of setting the hyperparameters within a family of covariance functions as well as comparing them across different families. Training of a GP entails the selection of a covariance function and its hyperparameters. This definition of training is in contrast to its use in Support Vector Machine (SVM) literature where the term refers to finding the support vectors for a fixed kernel.

For example, the squared exponential (SE) covariance function can be parameterized in terms of hyperparameters,

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp\left\{-\frac{1}{2}(\boldsymbol{x}_p - \boldsymbol{x}_q)^T M(\boldsymbol{x}_p - \boldsymbol{x}_q)\right\} + \sigma_n^2 \delta_{pq} \tag{103}$$

where $\boldsymbol{\theta} = (\{\boldsymbol{M}\}, \sigma_f^2, \sigma_n^2)$ is a vector containing all the hyperparameters, in which M denotes the parameters in the symmetric matrix M. The noise level parameter σ_n^2 is treated in a similar manner as a hyperparameter, so it may be, for all intents and purposes, be considered as such. Possible choices for the matrix M include

$$M_1 = l^{-2} I \tag{104}$$

$$M_2 = \mathrm{diag}(\boldsymbol{l})^{-2} \tag{105}$$

$$M_3 = \Lambda\Lambda^T + \mathrm{diag}(\boldsymbol{l})^{-2} \tag{106}$$

where l is a vector of positive values, and Λ is a $D \times k$ matrix, $k < D$. For many covariance functions it is relatively straightforward to interpret the meaning of its hyperparameters which is of great importance when it comes to understanding the experimental data. In the case of the SE covariance function with distance measure M_2, the $l_1, \dots, l_D$ hyperparameters represent the characteristic length-scales, that can be described as the distance along a particular axis in input space such that the function values become uncorrelated. Such a covariance function implements automatic relevance determination (ARD), since the inverse of the length-scale determines the relevance of a particular input. If the value of the length-scale is very large, the covariance will become almost independent of that input, effectively removing it from the inference [45]. There is plenty of slack for variation even within a family of covariance functions and the task is, based on a set of training data, to make inferences about the form and parameters of the covariance function, or equivalently, about the relationships in the data.

The task of model selection is essentially open-ended. Even for the SE covariance function, the is a large variety of possible distance measures, enabling the true learning from data. This variety necessitates a systematic and practical approach to model selection. Concisely, we need to be able to compare two or more methods whose values of particular parameters differ or compare the performance of a GP to the performance of any other model. There are many suggestions for model selection in the literature. However, there are three general principles that need to be observed: (1) compute the probability of the model given the data, (2) estimate the generalization error, and (3) bound a generalization error. The term generalization error refers to the average error on unseen test examples from the same distribution. The training error is not a useful measure of how the model will generalize to unseen data because the model may fit the noise in the training set, leading to a low training error but poor generalization performance. Poor generalization performance is realized when the training error is low, but the generalization error is significantly larger. This concept is also referred to overfitting the data. The next subsection covers the Bayesian view on model selection, which involves the computation of the probability of the model, given the data [45].

Chapter 3-4-5: Bayesian Model Selection

The computations that are needed for Bayesian inference are based on the posterior over the parameters, represented by $\boldsymbol{w}$ at the lowest levels, which is computed via Bayes' rule

$$p(\boldsymbol{w}|\boldsymbol{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\boldsymbol{y}|X,\boldsymbol{w},\mathcal{H}_i)p(\boldsymbol{w}|\boldsymbol{\theta},\mathcal{H}_i)}{p(\boldsymbol{y}|X,\boldsymbol{\theta},\mathcal{H}_i)}, \tag{107}$$

where $p(\boldsymbol{y}|X, \boldsymbol{w}, \mathcal{H}_i)$ is the likelihood and $p(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{H}_i)$ is the parameter prior. The prior encodes our knowledge about the parameters before seeing the data in the form of a probability distribution. If this prior knowledge about the parameters is vague, then the prior distribution is chosen to be broad. Thus, the posterior combines the information from the prior with the data, through the likelihood. The normalizing constant $p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)$ is independent of the parameters and it is referred to as the marginal likelihood or evidence, and is computed by

$$p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\boldsymbol{y}|X, \boldsymbol{w}, \mathcal{H}_i)\, p(\boldsymbol{w}|\boldsymbol{\theta}, \mathcal{H}_i) d\boldsymbol{w}. \tag{108}$$

Next, the posterior over the parameters, in which the marginal likelihood from above plays the role of the likelihood, is computed by

$$p(\boldsymbol{\theta}|\boldsymbol{y}, X, \mathcal{H}_i) = \frac{p(\boldsymbol{y}|X,\boldsymbol{\theta},\mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)}{p(\boldsymbol{y}|X,\mathcal{H}_i)}, \tag{109}$$

where $p(\boldsymbol{\theta}|\mathcal{H}_i)$ is the hyper-prior and where the normalizing constant is given by

$$p(\boldsymbol{y}|X, \mathcal{H}_i) = \int p(\boldsymbol{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)\, p(\boldsymbol{\theta}|\mathcal{H}_i) d\boldsymbol{\theta}. \tag{110}$$

Finally, the posterior of the model given the data is computed by

$$p(\mathcal{H}_i|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X,\mathcal{H}_i)p(\mathcal{H}_i)}{p(\boldsymbol{y}|X)}, \tag{111}$$

where the normalizing constant $p(\boldsymbol{y}|X) = \sum_i p(\boldsymbol{y}|X, \mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)$. Note that the implementation of Bayesian inference calls for the evaluation of several, possibly intractable integrals that may call for the application of one of the approximation methods such as Markov chain or Monte Carlo (MCMC) [45].

Chapter 3-4-6: Cross-validation

Another method for model selection is cross-validation (CV). The idea behind CV is to split the training data set into two disjoint sets, one of which is used for training, and the other is used to validate model performance as a proxy for the generalization error. Model selection using CV is carried out using the measurement of the generalization error. A negative consequence of the holdout method is that only a fraction of the data can be used for training, and if the training data set is too small, it can lead to a poor generalization performance of the model. K-fold cross-validation is used to resolve this problem. In a k-fold CV setting, the data is split into k disjoint, equally sized subsets. Following the split, validation is performed on a single subset and training is done using the union of the remaining $k-1$ subsets. The entire process is repeated k times, each time with a different subset for validation. Using the k-fold CV process, a large fraction of the data can be used for training and all cases appear as validation cases. The price for the robustness of validation is paid by computational time, because we are training k models instead of one. Typical values of k range from 3 to 10, with 10 being the gold standard [45].

Chapter 3-4-7: Model Selection for GP Classification

This subsection deals with the computation of derivatives of the approximate marginal likelihood for the Laplace and expectation propagation (EP) method discussed earlier. The section also presents algorithms for these cases and discusses the possible use of CV for training binary GP classifiers, among other methods.

The approximate log marginal likelihood is given by

$$\log q\,(\boldsymbol{y}|X,\boldsymbol{\theta}) = -\tfrac{1}{2}\hat{\boldsymbol{f}}^T K^{-1}\hat{\boldsymbol{f}} + \log p\big(\boldsymbol{y}|\hat{\boldsymbol{f}}\big) - \tfrac{1}{2}\log|B|, \qquad (112)$$

where $B = I + W^{\frac{1}{2}}KW^{\frac{1}{2}}$ and $\hat{\boldsymbol{f}}$ is the maximum of the posterior determined using Newton's method, and W is the diagonal matrix $W = -\nabla\nabla \log p\big(\boldsymbol{y}|\hat{\boldsymbol{f}}\big)$. The optimization of the approximate marginal likelihood $q(\boldsymbol{y}|X,\boldsymbol{\theta})$ with respect to the hyperparameters $\boldsymbol{\theta}$**.** To determine the, it is necessary to obtain the partial derivative $\partial(\boldsymbol{y}|X,\boldsymbol{\theta})/\partial\,\theta_j$, and with the covariance matrix K, $\hat{\boldsymbol{f}}$**,** and W all being a function of the hyperparameters $\boldsymbol{\theta}$**,** we obtain

$$\frac{\partial \log q(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial\theta_j} = \left.\frac{\partial \log q(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial\theta_j}\right|_{explicit} + \sum_{i=1}^{n}\frac{\partial \log q(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial \hat{f}_j}\frac{\partial \hat{f}_i}{\partial\theta_j}, \qquad (113)$$

where

$$\left.\frac{\partial \log q(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial \theta_j}\right|_{explicit} = \frac{1}{2}\hat{\boldsymbol{f}}^T K^{-1} \frac{\partial K}{\partial \theta_j} K^{-1}\hat{\boldsymbol{f}} - \frac{1}{2} tr\left((W^{-1}+K)^{-1}\frac{\partial K}{\partial \theta_j}\right), \quad (114)$$

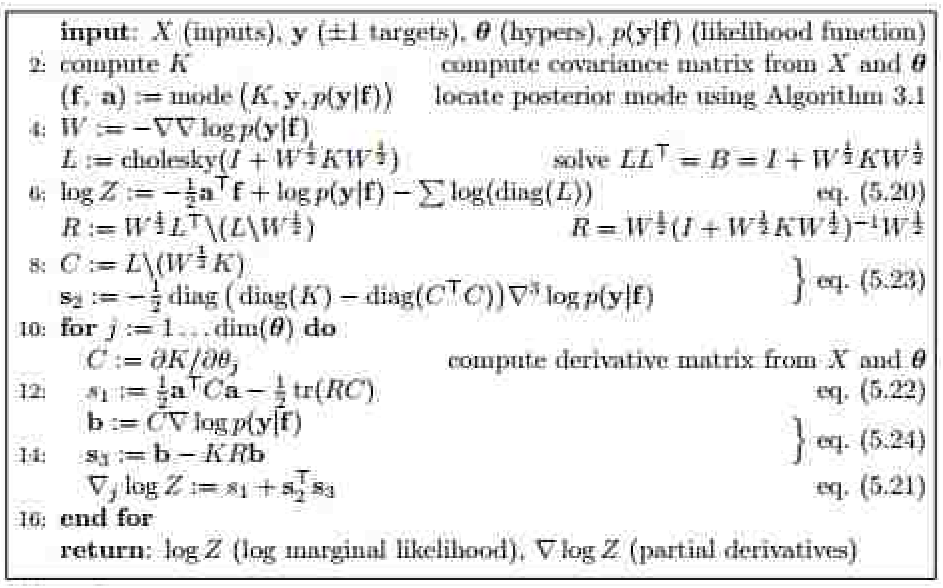
and where

$$\begin{aligned}\frac{\partial \log q(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial \hat{f}_i} &= -\frac{1}{2}\frac{\partial \log|B|}{\partial \hat{f}_i} = -\frac{1}{2} tr\left((K^{-1}+W)^{-1}\frac{\partial W}{\partial \hat{f}_i}\right) \\ &= -\frac{1}{2}[(K^{-1}+W)^{-1}]_{ii}\frac{\partial^3}{\partial f_i^3}\log p(\boldsymbol{y}|\hat{\boldsymbol{f}}),\end{aligned} \quad (115)$$

and

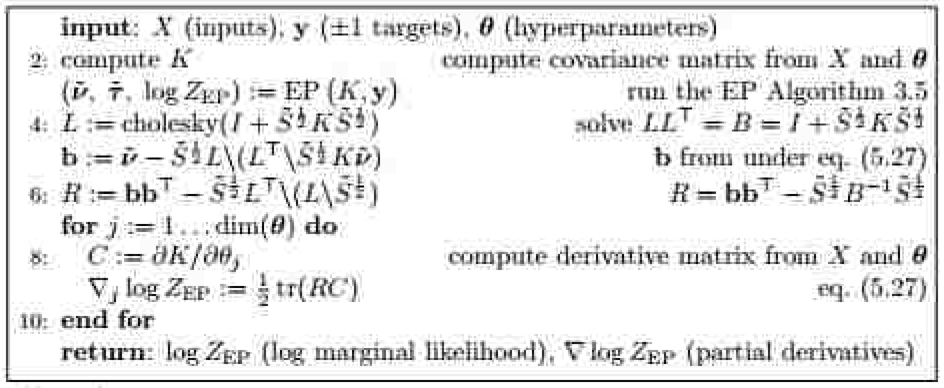
$$\begin{aligned}\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} &= \frac{\partial K}{\partial \theta_j}\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) + K\frac{\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}})}{\partial \hat{\boldsymbol{f}}}\frac{\partial \hat{\boldsymbol{f}}}{\partial \theta_j} \\ &= (I+KW)^{-1}\frac{\partial K}{\partial \theta_j}\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}).\end{aligned} \quad (116)$$

Fig. 45 shows the implementation of the GP classification in which the normalizing integral is performed using the Laplace approximation technique. Deemed a better alternative to the Laplace approximation, EP will be used to analyze MCU upset data. Fig. 46 summarizes the implementation of the EP method.



```
input: $X$ (inputs), $\mathbf{y}$ ($\pm 1$ targets), $\boldsymbol{\theta}$ (hypers), $p(\mathbf{y}|\mathbf{f})$ (likelihood function)
compute $K$                                           compute covariance matrix from $X$ and $\boldsymbol{\theta}$
$(\mathbf{f}, \mathbf{a}) := \text{mode}\left(K, \mathbf{y}, p(\mathbf{y}|\mathbf{f})\right)$      locate posterior mode using Algorithm 3.1
$W := -\nabla\nabla \log p(\mathbf{y}|\mathbf{f})$
$L := \text{cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$          solve $LL^\top = B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$
$\log Z := -\frac{1}{2}\mathbf{a}^\top \mathbf{f} + \log p(\mathbf{y}|\mathbf{f}) - \sum \log(\text{diag}(L))$      eq. (5.20)
$R := W^{\frac{1}{2}} L^\top \backslash (L \backslash W^{\frac{1}{2}})$          $R = W^{\frac{1}{2}}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})^{-1} W^{\frac{1}{2}}$
$C := L \backslash (W^{\frac{1}{2}} K)$                                      } eq. (5.23)
$\mathbf{s}_2 := -\frac{1}{2}\text{diag}\left(\text{diag}(K) - \text{diag}(C^\top C)\right)\nabla^3 \log p(\mathbf{y}|\mathbf{f})$    }
for $j := 1 \dots \dim(\boldsymbol{\theta})$ do
    $C := \partial K / \partial \theta_j$                         compute derivative matrix from $X$ and $\boldsymbol{\theta}$
    $s_1 := \frac{1}{2}\mathbf{a}^\top C \mathbf{a} - \frac{1}{2}\text{tr}(RC)$                     eq. (5.22)
    $\mathbf{b} := C \nabla \log p(\mathbf{y}|\mathbf{f})$                          } eq. (5.24)
    $\mathbf{s}_3 := \mathbf{b} - KR\mathbf{b}$                                  }
    $\nabla_j \log Z := s_1 + \mathbf{s}_2^\top \mathbf{s}_3$                          eq. (5.21)
end for
return: $\log Z$ (log marginal likelihood), $\nabla \log Z$ (partial derivatives)
```

Figure 45: GPC with Laplace's approximation for computation of log marginal likelihood and derivatives [45].



```
input: $X$ (inputs), $\mathbf{y}$ ($\pm 1$ targets), $\boldsymbol{\theta}$ (hyperparameters)
compute $K$                                           compute covariance matrix from $X$ and $\boldsymbol{\theta}$
$(\tilde{\boldsymbol{\nu}}, \tilde{\boldsymbol{\tau}}, \log Z_{\text{EP}}) := \text{EP}\left(K, \mathbf{y}\right)$                    run the EP Algorithm 3.5
$L := \text{cholesky}(I + \tilde{S}^{\frac{1}{2}} K \tilde{S}^{\frac{1}{2}})$              solve $LL^\top = B = I + \tilde{S}^{\frac{1}{2}} K \tilde{S}^{\frac{1}{2}}$
$\mathbf{b} := \tilde{\boldsymbol{\nu}} - \tilde{S}^{\frac{1}{2}} L \backslash (L^\top \backslash \tilde{S}^{\frac{1}{2}} K \tilde{\boldsymbol{\nu}})$              $\mathbf{b}$ from under eq. (5.27)
$R := \mathbf{b}\mathbf{b}^\top - \tilde{S}^{\frac{1}{2}} L^\top \backslash (L \backslash \tilde{S}^{\frac{1}{2}})$               $R = \mathbf{b}\mathbf{b}^\top - \tilde{S}^{\frac{1}{2}} B^{-1} \tilde{S}^{\frac{1}{2}}$
for $j := 1 \dots \dim(\boldsymbol{\theta})$ do
    $C := \partial K / \partial \theta_j$                         compute derivative matrix from $X$ and $\boldsymbol{\theta}$
    $\nabla_j \log Z_{\text{EP}} := \frac{1}{2}\text{tr}(RC)$                             eq. (5.27)
end for
return: $\log Z_{\text{EP}}$ (log marginal likelihood), $\nabla \log Z_{\text{EP}}$ (partial derivatives)
```

Figure 46: GPC with EP approximation for the computation of marginal likelihood [45].

Chapter 3-4-8: Approximation for Large Datasets

The most significant problem with GP prediction is that it is computationally expensive, as it scales with $\mathcal{O}(n^3)$, making the storage of the Gram matrix and solving of the associated linear systems prohibitive on modern workstations for data sets that contain $n > 10{,}000$ data points on most modern workstations. This boundary can be stretched further by using high-performance machines of supercomputers. The approximation methods for GPC are similar to those for GPR, but they need to deal with the non-Gaussian likelihood as well, either by using the Laplace approximation or expectation propagation (EP), which lends itself to sparsity. The three most widely used methods for approximations for large datasets in GPs are the subset of regressors (SR), subset of datapoints (SD), the projected process (PP) approximation, and the Bayesian committee machine (BCM).

For the SR method, we can use the model $f_{SR}(x_*) = \sum_{i=1}^{m} \alpha_i k(x_*, x_i)$ with $\boldsymbol{\alpha_m} \sim \mathcal{N}(\mathbf{0}, K_{mm}^{-1})$. The likelihood is non-Gaussian, but the optimization problem to find the MAP value of $\boldsymbol{\alpha_m}$ is convex and can be obtain via the Newton iteration method. Using the MAP value of $\boldsymbol{\alpha_m}$ and the Hessian at this point we can obtain a predictive mean and variance of $f(x_*)$ and obtain a probabilistic prediction by feeding it through a sigmoid. The choice of which subset of points to use can be done through a clustering method [50] or a forward selection strategy [51].

Another method is to select a subset of data points (SD) [52], using an EP-style approximation of the posterior, and the differential entropy score to select new points for inclusion. The EP-approach naturally encompasses sparsification. The projected process (PP) approximation can also be used with non-Gaussian likelihoods via an online method where samples are processed sequentially. This method also encompasses an expectation-propagation type algorithm where multiple sweeps through the data are allowed [53].

Finally, the BCM has also been generalized to accommodate non-Gaussian likelihood required for GPC [54]. Here, the dataset it broken up into blocks, but now approximate inference is carried out using the Laplace approximation in each block to yield an approximate predictive mean and an approximate predictive covariance, which are then combined to produce predictive probabilities.

For this research, a sample set of 3,000 training data points was sufficient to build a good model that approximates the PoE curve well, and the only sparse approximation used was expectation propagation. Having introduced the three machine learning methods (SVM, ANN, GP), the next section discusses the experiment that will be used to collect MCU upset data.

Chapter 4: Experiment Setup

The experimental procedure proposed to collect the data representing the phenomenon of MCU, and that will be used to validate the machine learning ability to predict MCU upsets is shown in Fig. 47.

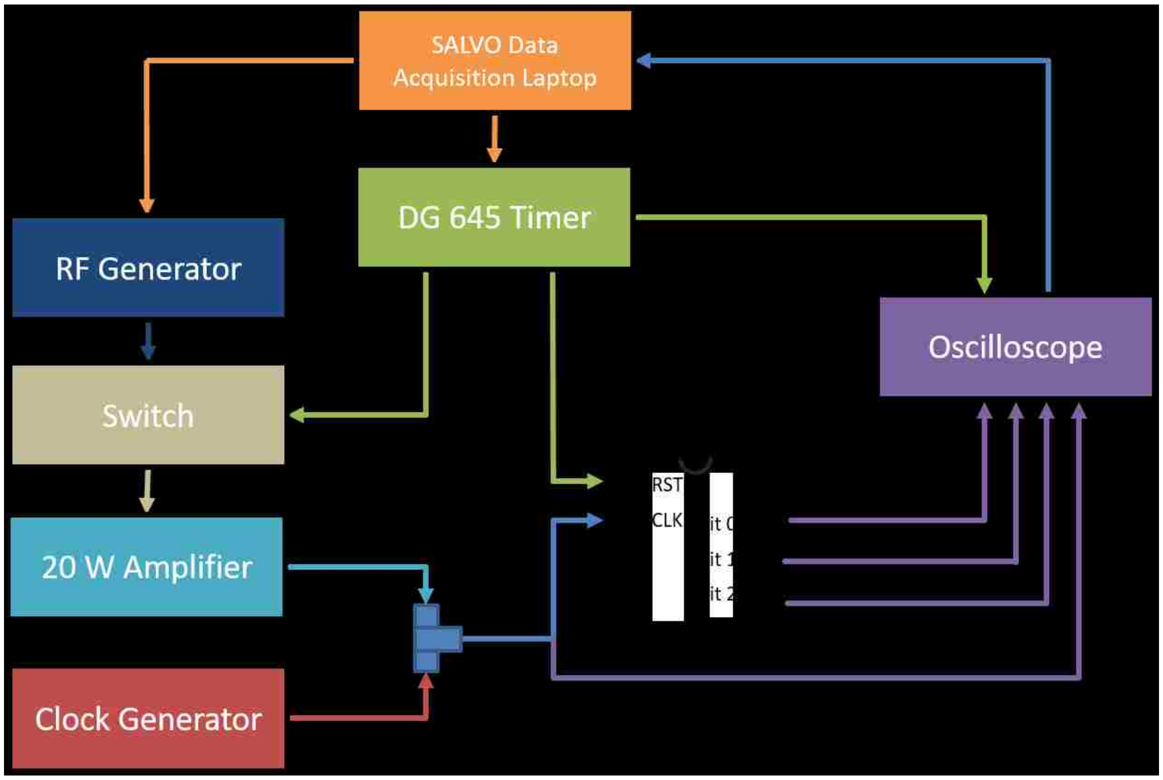


Figure 47: MCU upset experiment logical diagram.

The physical realization of this setup is shown in Fig. 48.

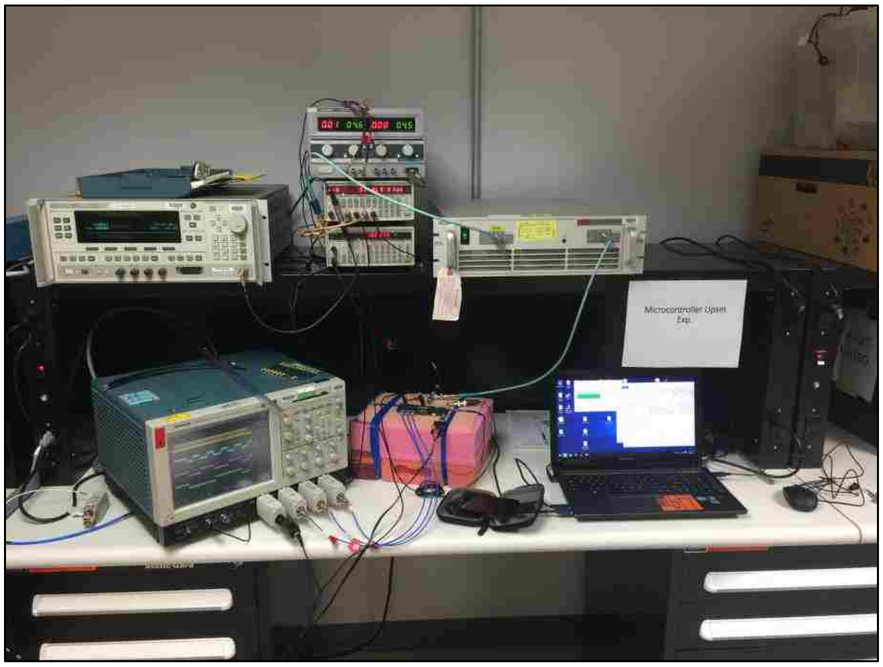


Figure 48: MCU Upset experiment setup.

Fig. 49 shows a close-up of the printed circuit board with the MCU and associated injection network.

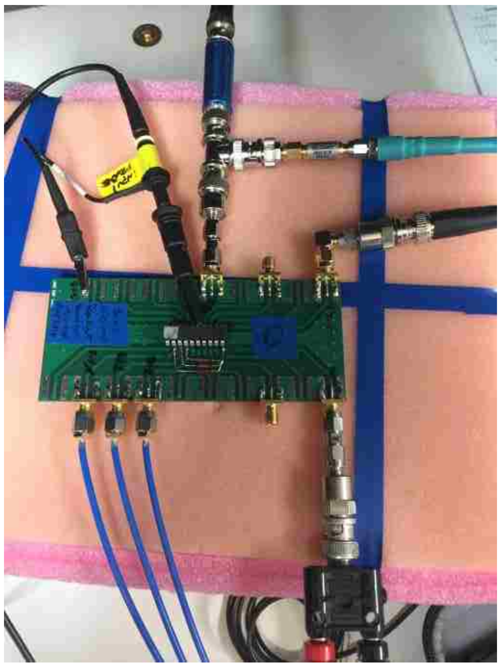


Figure 49: MCU with injection network.

The setup consists of:

- Stanford Research DG345 digital delay/pulse generator
- Stanford Research DG645 digital delay/pulse generator
- HP83620A synthesized sweeper
- Tektronix TDS7404 oscilloscope
- power supplies and amplifiers
- high impedance buffer amplifiers
- Tektronix current probe
- various high / low-pass filters.

The DS345 function generator is a 30 MHz synthesized function generator that generates many standard waveforms with excellent frequency resolution (1 μHz) and has flexible modulation capabilities including AM, FM, Burst, PM and frequency sweeps. The unit can also generate arbitrary waveforms as needed. Additionally, the DS345 has the capability to generate sinusoidal and square waves at frequencies up to 30.2 MHz, as well as triangle and ramp waveforms up to 100 kHz. The published frequency resolution for all of these functions is 1 μHz [55]. The main purpose of this device in the experimental setup is to provide the MCU clock signal.

The DG645 is a digital delay/pulse generator that provides precisely defined pulses at repetition rates up to 10 MHz. The instrument features low jitter, higher accuracy, faster trigger rates, and more outputs. The DG645 also features Ethernet, GPIB, and RS-232 interfaces for computer or network control of the instrument. The purpose of the instrument is to provide timing control and reference to the other devices in the setup.

The Hewlett Packard (HP) 83620A synthesized sweeper is used to generate the RF signal at varying frequency that is injected together with the clock into the clock pin of the MCU. The power of the RF signal will vary between -10 dBm and -20 dBm.

The setup illustrated in Fig. 47 is controlled using a Matlab program, called SALVO, which acts as the interface between the user and the equipment. As the initial step, the user creates a configuration file and imports it into the SALVO control suite. The program interprets the configuration file and distributes the relevant parameters to each instrument via the GPIB interface. Once the equipment is properly initialized, the test control suite activates the DG-645. This activation signal is designated as the reference time-stamp t_0 and serves as a reference point to all other commands. The DG 645 then sends signals to reset the MCU, cue the oscilloscope to record and toggle the RF switch. When the switch is toggled ON, a continuous wave (CW) radio frequency (RF) signal is allowed to pass to the amplifier for the duration of the user-defined pulse width, which results in the RF pulse being injected into the MCU clock pin, together with the clock signal. The MCU output bit values and the input clock/RF signals are recorded by the oscilloscope and retrieved by the SALVO data acquisition suite in Tektronix proprietary format. Data post-processing, any applicable visualization, and initial/manual analysis is performed using SALVO and other MATLAB-based scripts, respectively.

The parameters that are passed to the individual experiment components are controlled with a Microsoft Excel-based setup file. The setup file is designed to be able to specify experiment parameters such as injected RF pulse frequency, RF power, RF pulse duration, RF injection location on the clock signal, among others. Additionally, the setup file has the capability to accommodate these parameters for two district RF pulses that can be injected at specified locations with respect to the clock signal. For this research, only a single RF pulse injection will be used. After the brief overview of the setup file, a more detailed description is presented next.

The Excel-based setup file contains 21 experiment descriptors, some of which are control parameters such as RF pulse duration, RF power, RF frequency, and RF injection location. The first column in the setup file labeled "**Sequence #**" denotes the sequence number or also referred to as the experiment number. Each sequence can be thought of as a separate experiment in which one variable is modified and whose results (upset/no upset) are recorded by the oscilloscope, along with RF pulse parameters that are recorded at the DUT. The second column, labeled **"#Shots**" specifies the number of shot for a particular experiment or sequence, in other words, is specifies the number of times each sequence should be executed. The term "shot" signifies a single injection of RF pulse in the DUT. These two columns are used for internal software control purposes.

Next, the column termed "**Frequency_Hz**" denotes the frequency of the injected RF pulse in units of Hertz (Hz). This parameter is passed by SALVO to the RF sweeper instructing it to set the frequency of the RF pulse to the desired value. The fourth column termed "**Pwr_dBm**" denotes the power level of the injected RF pulse in dBm. These two setup parameters are examples of direct experiment parameters as opposed to internal control parameters used by the SALVO experiment control suite. The next four columns are used to define the location with respect to the clock signal as well the duration of the RF pulse, respectively.

The next four columns are termed "**P1_A_Ref**," "**P1_A_Dly**," "**P1_B Ref**," and "**P1_B_Dly**." Most references termed **"Ref"** are set to refer to **"T0"**, a reference set to the beginning of the experiment, or 0 nanoseconds. In this case "**P1_A_Ref**" and "**P1_B_Ref**" are both set to **"T0"**, whereas "**P1_A_Dly**" and "**P1_B_Dly**" refer to RF pulse injection location and RF pulse duration for the first pulse, respectively. An identical set of columns exists to describe characteristics for the second injected RF pulse, however, for this research, only the effects of one pulse will be modeled. Future research may include the additional pulse characterization. The last 14 columns are actual measured quantities relating to pulse width and magnitude, and energy delivered to the MCU clock pin, as well as their average and standard deviations.

During initial tests, the MCU that will be used as the DUT is an ATMEL AT89LP2052. This specific DUT is the first one to be tested, and it will be labeled DUT#1. Most of the description of this chip follows Atmel's datasheet. DUT#1 is a low-power, high-performance CMOS 8-bit microcontroller with 2Kbytes of programmable flash memory. DUT#1 fetches a single byte of memory every clock cycle. Traditionally, in the 8051 architecture, each fetch requires 6 clock cycles, forcing instructions to execute in 12,24, or 68 clock cycles. With this fetch feature, DUT#1 executes instructions in 1-4 clock cycles, thus greatly increasing throughput. Approximately 70% of instructions require only as many clock cycles as the number of bytes they execute and most of the remaining instructions only require one additional clock cycle. Fig. 50 shows a diagram of DUT#1 and its pin labels [56].

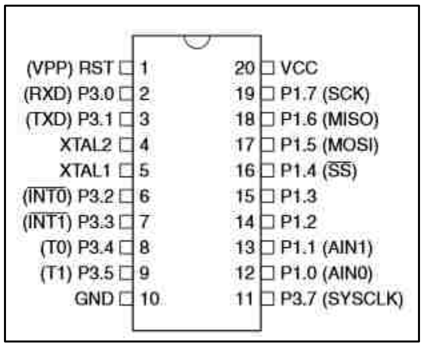


Figure 50: DUT#1 (ATMEL AT89LP2052) pin diagram [56].

DUT#1 uses a Harvard Architecture with separate address spaces for program memory and data memory. This DUT is part of a family of devices that are fully binary compatible with the MCS-51 instruction set. Even though there are significant compatibilities with existing standards, DUT#1 inherent differences in system behavior are a direct result of its high-performance nature. Details of specific differences in comparison to Standard 8051 are listed in the datasheet provided by the manufacturer. Fig. 51 shows a high-level overview of the underlying MCU architecture.

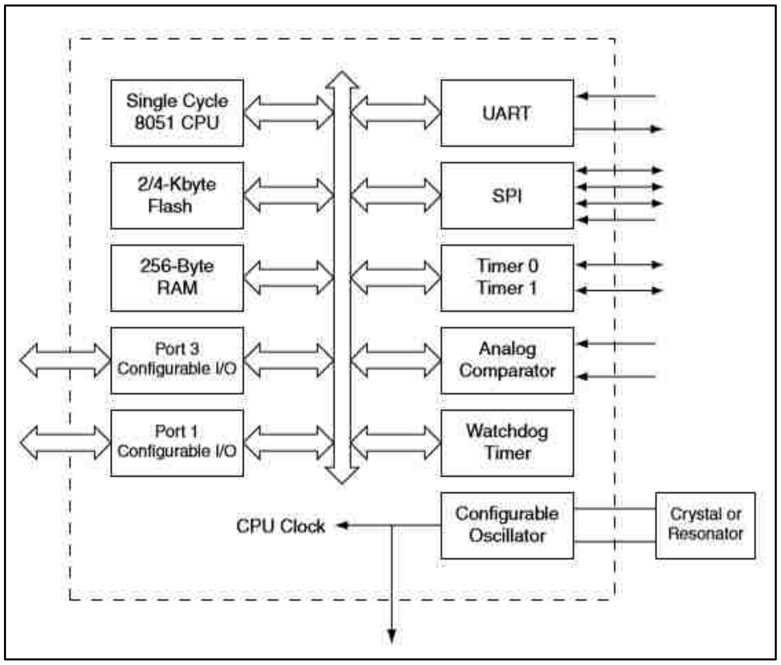


Figure 51: High-level architecture of DUT#1 [56].

DUT#1 possesses an enhanced CPU that runs 6-12 times faster than the standard 8051 devices. The faster performance can be attributed to two factors: first, the CPU fetches one instruction byte from the code memory every clock cycle and second, the CPU uses a simple two-stage pipeline to fetch and execute instructions in parallel. Fig. 52 shows a sample instruction execution diagram [56].

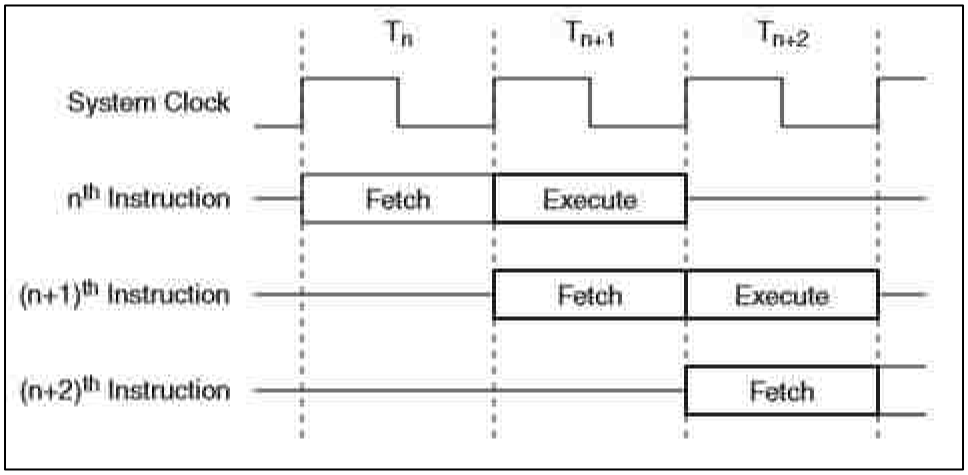


Figure 52: DUT#1 instruction execution timing diagram [56].

The main component of interest to this research is the system clock because that will be the periphery into which the RF signal is injected. The system clock on DUT#1 is generated directly from one of two selectable clock sources - the on-chip crystal oscillator and the external clock source. Only one of these options can be selected as the clock source at any given time. When enabled, the internal inverting oscillator amplifier is connected between XTAL1 and XTAL2 for connection to an external quartz or ceramic resonator. When a crystal oscillator is used, XTAL2 should not be used to drive a board-level clock. The external clock option is selected by setting the oscillator bypass fuse, disabling the amplifier and allowing XTAL1 to be driven directly by the clock source. For this experiment, an external clock source, specifically the DS345, is used [56].

The program that this MCU will execute is a simple counter procedure during which the MCU performs counting operations starting from 0 to 7 at a clock frequency of 1 MHz as shown in Table 9. The specific assembly-level commands are shown together with the amount of time in clock cycles that it takes the DUT#1 to execute the specific command. The overall time duration that it takes the DUT#1 to initialize and count from 0 to 7 is 95 μs. Because RF injection occurs at different locations on the clock signal, it is inherent that the location will correspond to a specific instruction being executed. Fig. 53 shows the output as recorded by the oscilloscope.

Table 9: Counter program in assembly language.

Instruction	Time (us)
NOP (1)	1-5 (1 clock cycle)
MOV SP,#080H (2)	6, 7 (2 clock cycles)
MOV 0C2H,#001H	8, 9 (2 clock cycles)
MOV 0C3H,#001H	10, 11 (2 clock cycles)
MOV P1,#001H	12, 13 (2 clock cycles)
L0011:	Loop Label (0 clock cycles)
MOV A,P1	14, 15 (2 clock cycles)
ADD A,#001H (3)	16, 17 (2 clock cycles)
MOV P1,A	18, 19 (2 clock cycles)
SJMP L0011 (4)	20, 21, 22 (3 cycles)

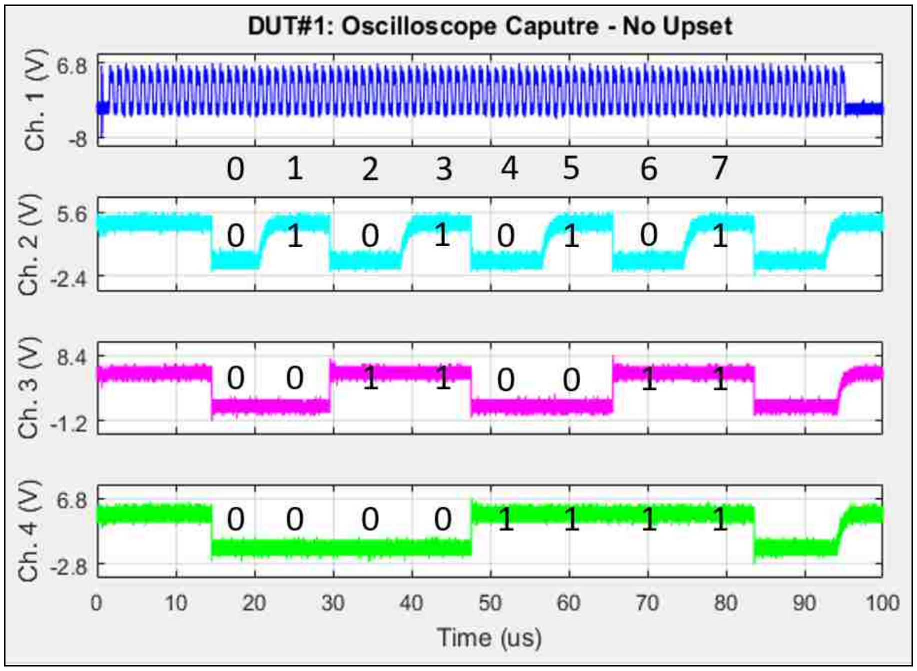


Figure 53: DUT#1 oscilloscope capture - normal operation.

Fig. 53 shows the output signals of the four oscilloscope channels. Channel 1 shows the clock signal as well as the injected RF pulse. Channels 2-4 show the output signals of the MCU output pins where channel 2,3, and 4 correspond to bits 0,1, and 2 of the 0-7 count, respectively. The row of decimal numbers between channel 1 and channel 2 represents the binary count translated to the decimal system. This experiment instance shows a microcontroller that is not affected by the injected RF pulse, in other words, this experiment yielded an outcome denoted as “no upset.”

“No Upset” denotes one of the binary outcomes of this experiment. The other binary outcome is denoted as “Upset,” and Fig. 54 shows the first example of an outcome with this label. If the expected bit value at a previously specified time is not a 0 or 1, the post-processing algorithm will label it as upset. Upset scenarios include the situation in which the bit value flips from an expected 0 to a 1, or vice versa. Additionally, it involves the scenario in which the bit shifts by a significant amount, i.e. more than a clock cycle to the left of right of the time at which it is expected to have a certain value. Even though the experiment exhibits several different types of upset, a binary label denoting normal operation or upset is utilized for the purpose of this research. Additional classifications of upset scenarios will be considered for future research.

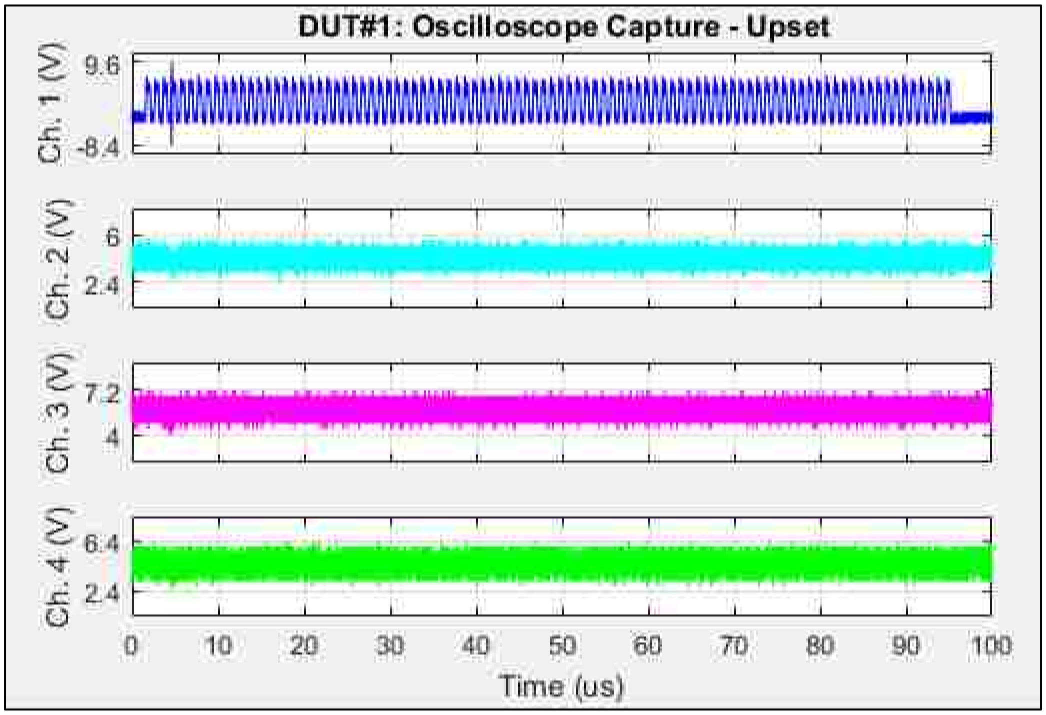


Figure 54: DUT#1 oscilloscope capture – upset.

The manifestation of MCU upset in Fig. 54 shows that the output pins representing bits 0,1, and 2 are in an unknown state, requiring a device reset to clear the condition. It can also be seen that the RF pulse injection location for this scenario differs that the injection location from that in Fig. 51.

Injecting RF pulses with different characteristics at different locations on the clock signal generates different types of upset scenarios. Fig. 55 shows an additional example of MCU upset. There are several more variations, but they will not be discussed in detail. Instead, the research will focus on a simplified type of analysis and prediction that is related to a binary experimental outcome – “upset” and “no upset.”

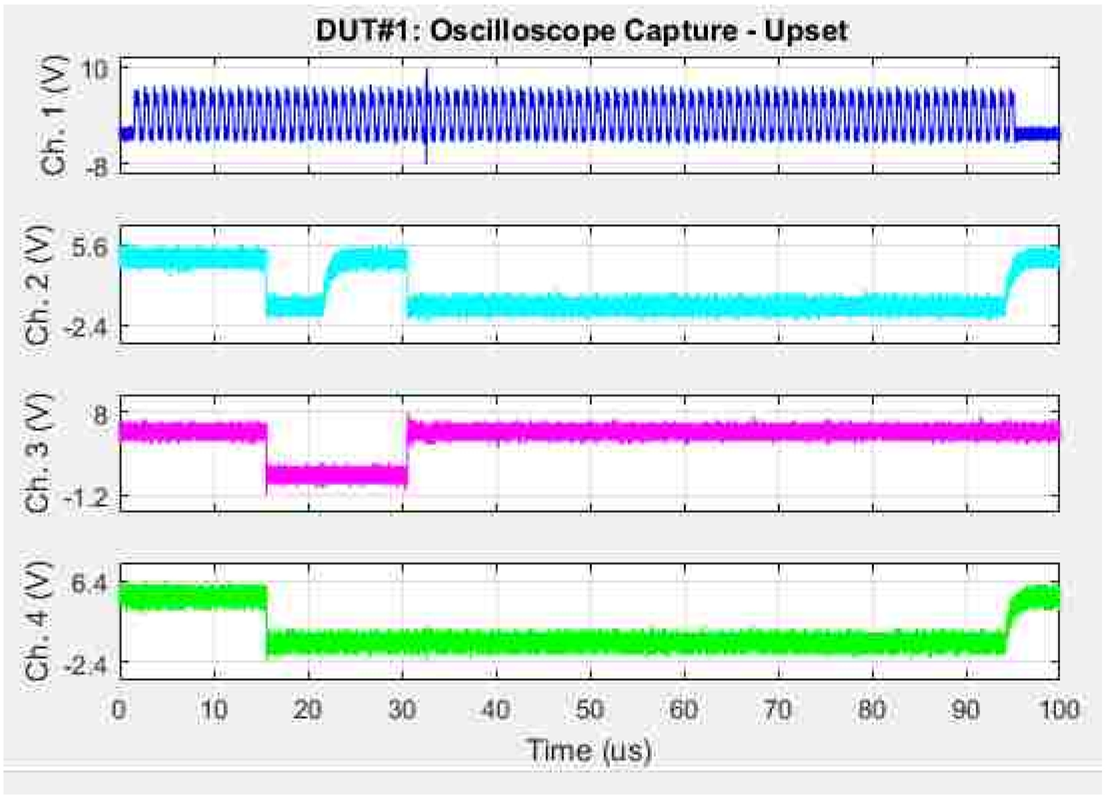


Figure 55: DUT#1 Oscilloscope Capture - Upset

During the data post-processing, several attributes about the RF pulse will be measured and recorded using the oscilloscope. Even though we request the RF sweeper to produce a perfect pulse at a given power and frequency, as that pulse propagates through the experiment setup, it will degrade and become noisy, causing slight signal variation at the physical point of injection, which is the clock pin and the subsequent signal processing physical architecture. During the course of this research, the effect of several changing variables on the software operation of the MCU has been studied. These variables and their ideal values are shown in Table 10.

Table 10: Variables studied during this research.

VARIABLE	VALUES UNDER CONSIDERATION (UNITS)
RF INJECTION LOCATION	Rising Clock Edge, Clock High, Falling Clock Edge, and Clock Low
RF SIGNAL DURATION	50, 100 (ns)
RF SIGNAL POWER LEVEL	-10,-15,-20 (dBm)
RF SIGNAL FREQUENCY	50, 100 (MHz)
MCU CLOCK FREQUENCY	1 (MHz)

These variable values represent the ideal case (instrument settings), and because of equipment idiosyncrasies and different types of noise present in the system, the quantities that are delivered to the MCU as measured by the oscilloscope may differ. The amount of EM energy coupled to the chip is also dependent on the internal chip state, as determined by the instruction being executed at the time.

Chapter 5: Experiment Data Analysis and Results

This chapter is organized into seven separate subsections. Section 5-1 describes the analysis strategy for the MCU upset problem. Sections 5-2 through 5-6 present analysis for the TE-1 through TE-5 test cases, respectively. The analysis of each test case is broken down to present machine learning results during training and during test. Section 5-7 discusses the results and provides a comparison between the three analysis methods and their effectiveness in tackling the MCU upset problem.

Chapter 5-1: Experiment Data Analysis Strategy

This section presents the key concepts behind MCU upset modeling using machine learning methods. During this research, the effect of several parameters (RF pulse width, RF injection location, RF power level, and RF frequency) on the MCU operation was studied. These variables and their nominal values are shown in Table 10 previously. The focus of this research is to study the effects of an interfering RF signal as it is injected into different stages of program execution. The dependent variable, in this case, is the event of upset and the cumulative probability of upset. The independent variables that affect the dependent variable are the various characteristics describing the nature of the RF signal and the injection location with respect to the MCU clock signal.

In this study, the MCU upset problem is formulated as a binary classification problem in machine learning. The two possible outcomes of this experiment are upset (1) or no upset (0 for SVMs and ANNs or -1 for GPs). A classification model (or classifier) represents a mapping from input vectors to predicted classes. Some classifiers produce a continuous output, such as an estimate of an instance’s class membership probability. In this research, NNs and GPMLS both produce a probability of upset, given the data, while

SVMs produce a discrete class label that indicates only the predicted class of each instance (1=upset/0=no upset). Various thresholding techniques can be applied to continuous outputs to predict class membership. To distinguish between the actual class and the predicted class, the labels $\{\boldsymbol{Y}, \boldsymbol{N}\}$ are used to denote class predictions produce by a given model. Given a classifier and an instance, there are four possible outcomes. If the instance is positive and the model classifies it as positive, it is counted as a true positive, and if it is classified as a negative, then it is counted as a false negative. Conversely, if the instance is negative and it is classified as negative, it is counted as a true negative, and if it is classified as positive, it is counted as a false positive. Given a classifier and a set of instances, a two-by-two confusion matrix (also referred to as a contingency table) can be constructed that represents the dispositions of the set of instances, as shown in Fig. 56.

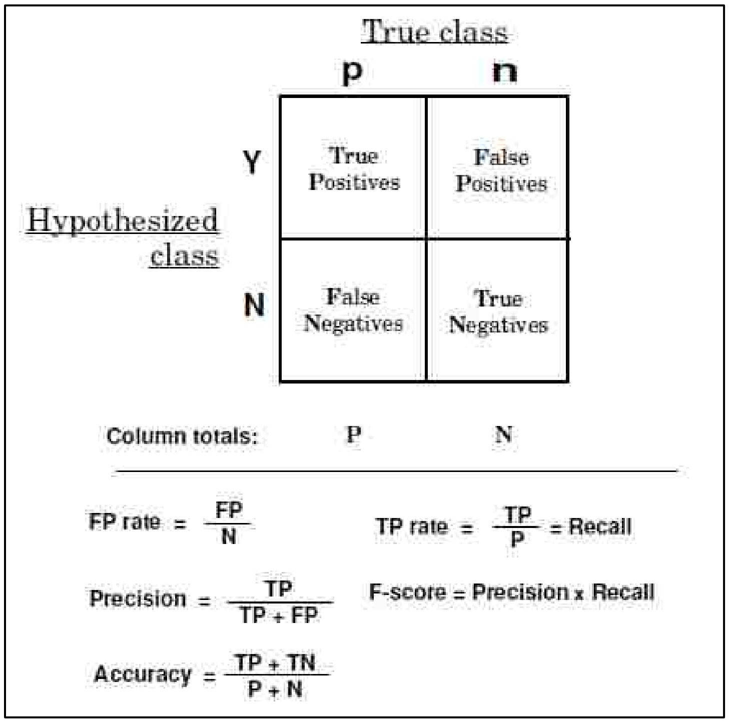


Figure 56: A confusion matrix and associated metrics [57].

The numbers along the major diagonal represent the correct classification made by the classifier, whereas the numbers off the diagonal represent errors – or confusion – between the classes. The confusion matrix forms the basis for many common metrics, such as the receiver operating characteristic (ROC) curve.

The ROC analysis and the confusion matrix will be used to assess the performance of a machine and its ability to predict MCU upset. ROC curves were developed by radar engineers during World War II for detecting enemy objects in the battlefield. They have also long been used in signal detection theory to depict the tradeoff between hit rates (signal) and false alarm rates (noise) of classifiers. In classification problems, such as the MCU upset binary classification problem, the ROC curve is a technique used to visualize, organize, and select classification models based on their performance. In statistics, the ROC curve is a graphical plot that is used to assess the performance of a binary classifier system as a function of its discrimination threshold.

Technically, ROC curves, also known as ROC graphs, are two-dimensional plots in which the true-positive rate (TPR) is plotted on the Y-axis, and the false-positive rate is plotted on the X-axis. This way, an ROC graph depicts relative trade-offs between benefits (‘true positives’) and costs (‘false positives’) . The curve is created by plotting the TPR against the FPR for various threshold settings. The true-positive rate is also referred to as sensitivity, recall, or probability of detection in machine learning applications. A sensitivity of 100% means that the classifier recognizes all observed negative cases. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 − specificity). ROC analysis can be used to select optimal models and to discard suboptimal ones [58], [59].

Fig. 57 illustrates the ROC space. The diagonal red line signifies that any model positioned on it is no better than a random guess in a binary classification problem. Models that are in the upper half (model ‘C’ and model ‘A’) will perform better, while model ‘C’ will perform worse than a random guess (model ‘B’).

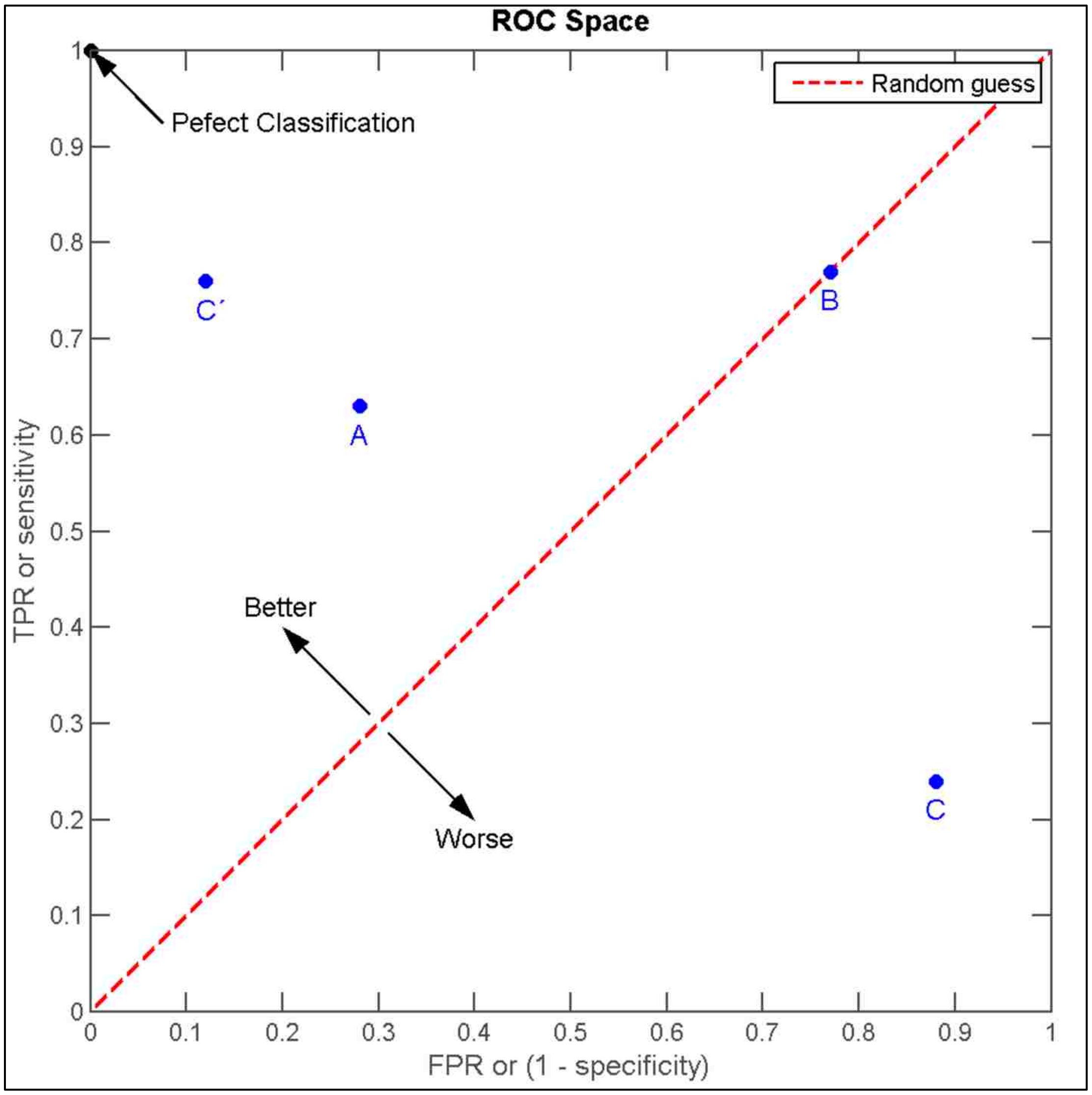


Figure 57: ROC space illustration [60].

In ROC analysis, the area under the curve (AUC) is a quantifier used to evaluate a model’s performance. An AUC of 1 signifies perfect classification, while an AUC of 0 signifies maximum misclassification. AUC=0.5 is as good as a random guess whether the injected RF signal will cause an MCU upset. Gronescu [61] gives the following rough guide for determining the accuracy of a classifier in terms of the AUC:

- $0.90 < AUC < 1.00 \rightarrow$ **excellent classification**
- $0.80 < AUC < 0.90 \rightarrow$ **good classification**
- $0.70 < AUC < 0.80 \rightarrow$ **fair classification**
- $0.60 < AUC < 0.70 \rightarrow$ **poor classification**
- $0.50 < AUC < 0.60 \rightarrow$ **failure**.

During the testing phase, the performance of the machine will be assessed and illustrated using a confusion matrix, shown in Fig. 56, ROC analysis as shown in Fig. 57, in addition to the overall PoE as a function of RF voltage and injection location with respect to the clock shown in the top plot of Fig. 58. The confusion matrix plot is used to understand how the classifier performed in each class and it helps to identify the areas where the classifier has performed poorly. The rows show the true class, and the columns show the predicted class. The diagonal cells show the cases in which the true class and predicted class match. If these cells are green, the classifier has performed well and classified observations of this true class correctly. A very interesting illustration that shows the suitability of each machine to model MCU upset will be the PoE vs. RF injection location plot for any given RF power level, RF frequency, and RF pulse duration as shown in the top plot of Fig. 58. A high prediction accuracy, as measured by the confusion matrix and an AUC close to 1 should translate to a close match between predicted and observed PoE curves on the PoE plot.

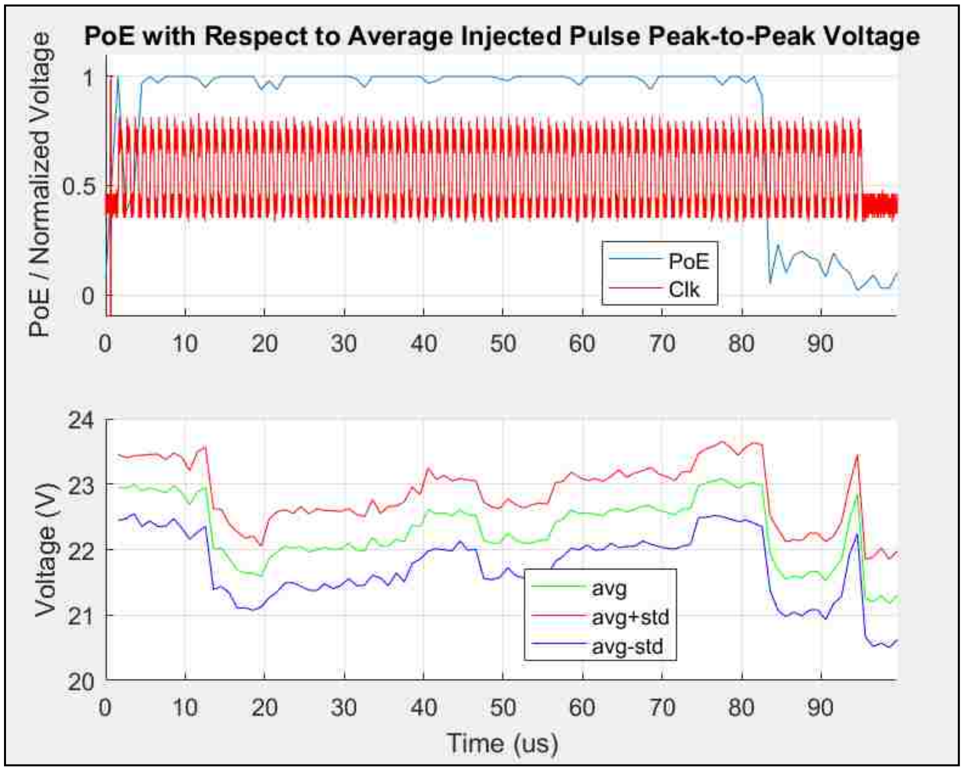


Figure 58: Probability of effect (PoE) as a function of injection location and RF voltage.

The top half of Fig. 58 shows how the PoE changes as a function of injection location with respect to time and the peak-to-peak voltage of the injected RF signal. In this case, the PoE is simply the frequency of effect at each injection location,

$$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}/_{\text{Number of Experiments}}\,.$$

Each machine learning technique (SVM, NN, GPML) will make a prediction (upset or no upset) for each of the test cases, and the predicted PoE would be generated using the equation above which in turn will be used to construct the PoE plot in Fig. 58. There are 5 test cases, each of which contains 38,300 training data points and 38,300 test data points. The training points are used to train the machine and select the best performing models, following which the machine will be tested on unseen test points. The training data set and the test data set both contain 38,300 data points for SVM and

NN analysis, while the GPML is constrained to 3000-5000 training data points to keep the computational complexity and training time reasonable. The performance of a machine to determine upset or no upset is evaluated during training and during test.

For the SVM training, there are two considerations that warrant extensive computing. First, the model parameters C (misclassification penalty) and σ (kernel scale) need to be determined, followed by the determination of support vectors. In the case of GPs, it should be noted that the computational complexity $[O(n^3)]$ during GPML training only allows for $n < 10{,}000$ training points [45] and thus only 3,000 - 5,000 points are used to determine GP hyperparameters. Specifically, 5,000 training points are chosen at random for the TE-1 data set, while 3,000 points are chosen at random for the remaining four data sets, for no other reason than to speed up GP training.

The ANN model has unknown parameters, often referred to as weights, the values of which need to be determined for the model to fit the training data well. For the classification case, the cross-entropy, or deviance, will be used to quantify ANN error during the training, validation, and test. The ANN that will be used is a two-layer feedforward neural network that contains sigmoid neurons in the hidden layer and softmax neurons in the output layer. Together with the softmax activation function and the cross-entropy error function, the ANN model is exactly a linear logistic regression model in the hidden units, and all its model parameters are estimated by maximum likelihood [36]. Typically, a global minimizer leads to an overfitted solution, and overfitting is prevented through a penalty term through early stopping of ANN training activities.

As with the SVM approach, all inputs for the ANN were standardized to have mean zero and a standard deviation of one, ensuring that all inputs are treated equally in the regularization process. Moreover, it is better to have too many hidden units that too few. A model that is built with too few hidden units may not have enough flexibility to capture the nonlinearities in the data, while too many units can have their weights shrunk to zero with the appropriate regularization technique. A typical range for the number of hidden units is somewhere in the range of 5 to 100. For this analysis, a neural network with 50 hidden units is used, and its architecture is summarized at a high level in Fig. 59. As with SVMs, the performance of the ANN will be evaluated during the training and test phases. Training of the ANN was achieved using the scaled conjugate gradient technique described earlier**.**

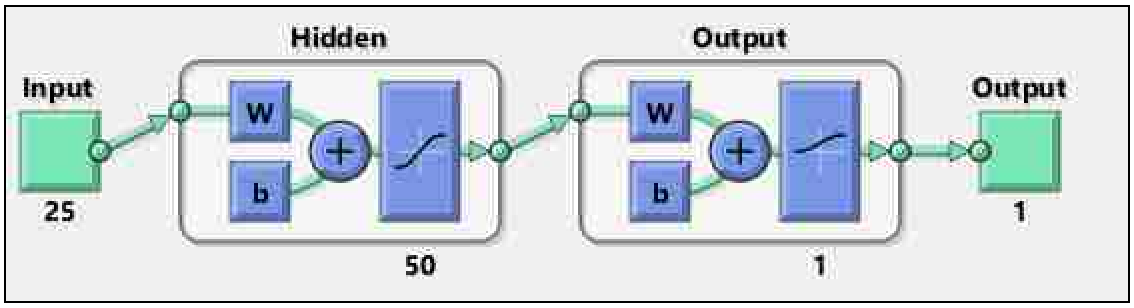


Figure 59: ANN architecture used to analyze MCU upset data.

There are several covariance function choices for GPML, with the squared exponential covariance function being one of the most widely used. Other functions include the polynomial covariance function of the second and third degree and the neural network, among many others. Each of these covariance functions typically has a number of free hyperparameters whose values need to be determined to assemble a complete GP model. The selection of a covariance function and its parameters is generally referred to as the training phase of a GP. The squared exponential covariance function is parameterized in terms of its hyperparameters as follows:

$$k(\boldsymbol{x_p}, \boldsymbol{x_q}) = \sigma_f^2 \exp\left(-\tfrac{1}{2}(\boldsymbol{x_p} - \boldsymbol{x_q})^T M(\boldsymbol{x_p} - \boldsymbol{x_q})\right) + \sigma_n^2 \delta_{pq} , \qquad (117)$$

with $\boldsymbol{\theta} = (\{M\}, \sigma_f^2, \sigma_n^2)$ being a vector that contains all the hyperparameter, and σ_n^2 denoting the noise level parameter. Possible choices for the matrix M are:

$$M_1 = l^{-2} I \qquad (118)$$

$$M_2 = \mathrm{diag}(l)^{-2} \qquad (119)$$

$$M_3 = \Lambda\Lambda^T + \mathrm{diag}(l)^{-2}, \qquad (120)$$

where l represents a vector of positive values and Λ is a $D \times k$ matrix, where $k < D$. The properties of covariance functions depend on the values of hyperparameters, which are determined from data. For the square exponential covariance function above with distance measure M_2, the $l_1, \dots, l_D$ hyperparameters play the role of characteristic length-scales, in other words, they encode how far you need to move along a particular axis in input space for the function values to become uncorrelated. Such a covariance function implements automatic relevance determination (ARD), which has been successfully used to remove irrelevant inputs [45]. The squared-exponential covariance function with ARD is used for training and testing GPs on MCU upset data as well.

The number of training data for the GP is not identical to the training data and composition of the SVM and the ANN. The reason for this is a significant problem with

GP prediction in that its computational complexity typically scales as $\mathcal{O}(n^3)$, resulting in large storage requirements for the Gram matrix and significant computational burden for solving the associated linear systems, although this restriction boundary could be stretched further by using supercomputers. Because of this limitation, a random set of 5,000 data points is chosen from the TRG-1 dataset and the hyperparameters are trained using the 5,000 points, and 3,000 points are chosen at random for the next four data sets. Testing of the GPML, however, is still done on the full 38,300 data points in all experiments.

Five experiments were executed as part of this research, each of which contained a set of 38,300 training points and 38,300 test points, for a total of 383,000 total points with the following experiment variables: injection times with respect to the clock signal, injected RF power, injected RF pulse width, and injected RF frequency. For the 191,500 training points, the average training error was 12.47%, while for the 191,500 test points the average test error was 14.85%, meaning that on average, the machine was able to predict MCU upset with an 85.15% accuracy during testing. Leaving out the results for the worst-performing model (SVM with a linear kernel), the test prediction accuracy for the remaining machines is almost 89%. All three machine learning methods (ANNs, SVMs, and GPML) showed excellent and consistent results in their ability to model and predict the PoE on an MCU due to IEMI, with the GP approach performing best during training with a 7.43% error, while the ANN technique was most accurate during test with a 10.80% error. The summary of results is shown in Table 11.

Table 11: Machine prediction of MCU upset - summary of results.

TE-ALL Data (TRG (N)=191,500 / TST (N)=191,500, TOTAL(N)=383,000) Injected RF Frequency $(f) = 50, 100\ MHz$, Injected RF Power $(P) = -10, -15, -20\ dBm$, Injected RF Pulse Width $(PW) = 50, 100\ ns$		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel	0.1544	0.1837
SVM – 2nd Degree Polynomial (Quadratic) Kernel	0.1018	0.1146
SVM – 3rd Degree Polynomial (Cubic) Kernel	0.1022	0.1170
SVM – Radial Basis Function (RBF/Gaussian) Kernel	0.0954	0.1086
Artificial Neural Network (ANN)	0.0952	0.1080
GPs with SE-ARD Covariance Function (EP) / likERF	0.0743	0.1104
GPs with SE-ARD Covariance Function (LP) / likERF	0.0888	0.1076
AVERAGE ERROR	**0.1247**	**0.1485**

Table 11 also summarizes the results for GPs that were obtained using the Laplace approximation (LP) method in addition to the EP method. Even though Rasmussen and Williams [47] present results that favor EP over LP, for this research, it appears that approximating the Bernoulli distribution using Gaussians via Laplace approximations yields slightly better results. Laplace approximation probabilistic output is within 1% when compared to the EP probabilistic output when they are compared against the ANN softmax probabilistic output, but the Laplace approximation method is significantly faster (minutes vs. hours) during GP training and is also qualitatively faster during prediction.

Chapter 5-2: Test Experiment 1 (TE-1) Data Analysis

The data for test experiment 1 (TE-1) is divided into two sets. The first set of 38,300 data points is used to train SVMs and the ANN and select a model that provides the best fit, given the training data (TRG-1 data). The GP is to be trained on a randomly selected subset of 5,000 data points from the TRG-1 data set. The second set consists of 38,300 test data points (TST-1 data). The TST-1 dataset will be used to test how well each machine generalizes to unseen data and to generate the PoE curves. None of the machines under consideration have been exposed to the TST-1 data set.

For the TE-1 data set, the injected RF frequency is $f = 100$ MHz, the injected RF power level is $P = -10$ dBm, and the injected RF pulse width is $PW = 100$ ns. The RF pulse is injected into the clock pin, and it is timed to coincide with each clock edge (rising edge, clock high, falling edge, clock low) starting at time $t = 0.5\ \mu s$ until $t = 95\ \mu s$, which is the time window during which the MCU initializes and completes the counting operation. In sum, there are 383 injection locations during the clock window, each consisting of 100 shots (instance during which RF is injected into the clock pin), for a total of 38,300 data points. The first test run is used to collect 38,300 data points that will be used to train each machine, except the GP, which will be trained with a randomly selected subset of 5,000 data points. The test is then repeated to collect an additional 38,300 data that will be used to test each machine.

The machines under consideration include SVMs with linear (SVM-LINEAR), quadratic (SVM-QUADRATIC), cubic (SVM-CUBIC), and Gaussian/radial basis function (SVM-RBF) kernels. The other two models used to model MCU upset include a two-layer feedforward neural network (ANN) that is trained using the scaled conjugate gradient and GPs with the squared exponential covariance functions with automatic relevance determination (GP-SE-ARD).

The performance of each machine is assessed during training and during the test phase. The training assessment features a Bayesian optimization analysis for the selection of key SVM parameters, a confusion matrix for training data, and ROC analysis for training data for all classifiers. The test assessment features a confusion matrix for test data, ROC analysis for test data, and a plot of the PoE as a function of injection location, injected RF power, injected RF frequency, and injected RF pulse duration for all classifiers.

Classifier Performance During Training – TRG-1 Data

In classification and regression scenarios, SVMs require that the kernel function (with its scale) be chosen along with the misclassification penalty parameter C. One of the more effective ways to do this is to utilize Bayesian optimization, as described in Section 3-5. The objective of Bayesian optimization here is to choose parameters C and σ that will minimize the cross-validation error. Even though four different kernel types were analyzed for the SVM models, only the two best performing SVM models will be presented in more detail for each test case.

For the first training case (TRG-1) as well as the corresponding test case (TST-1), both of which contain 38,300 test points, the best performing SVM kernel is the RBF / Gaussian kernel (SVM-RBF), while the second best SVM kernel is the third-degree polynomial (SVM-CUBIC). The selection of kernel parameters for all SVM kernels was done using a Bayesian optimization method that minimized the 10-fold cross-validated loss, which for the SVM-RBF resulted in $C = 3.3734,$ and $\sigma = 0.55027$ with a Bayes-estimated CV-error of 0.09684 (9.684%), as shown in Fig. 60.

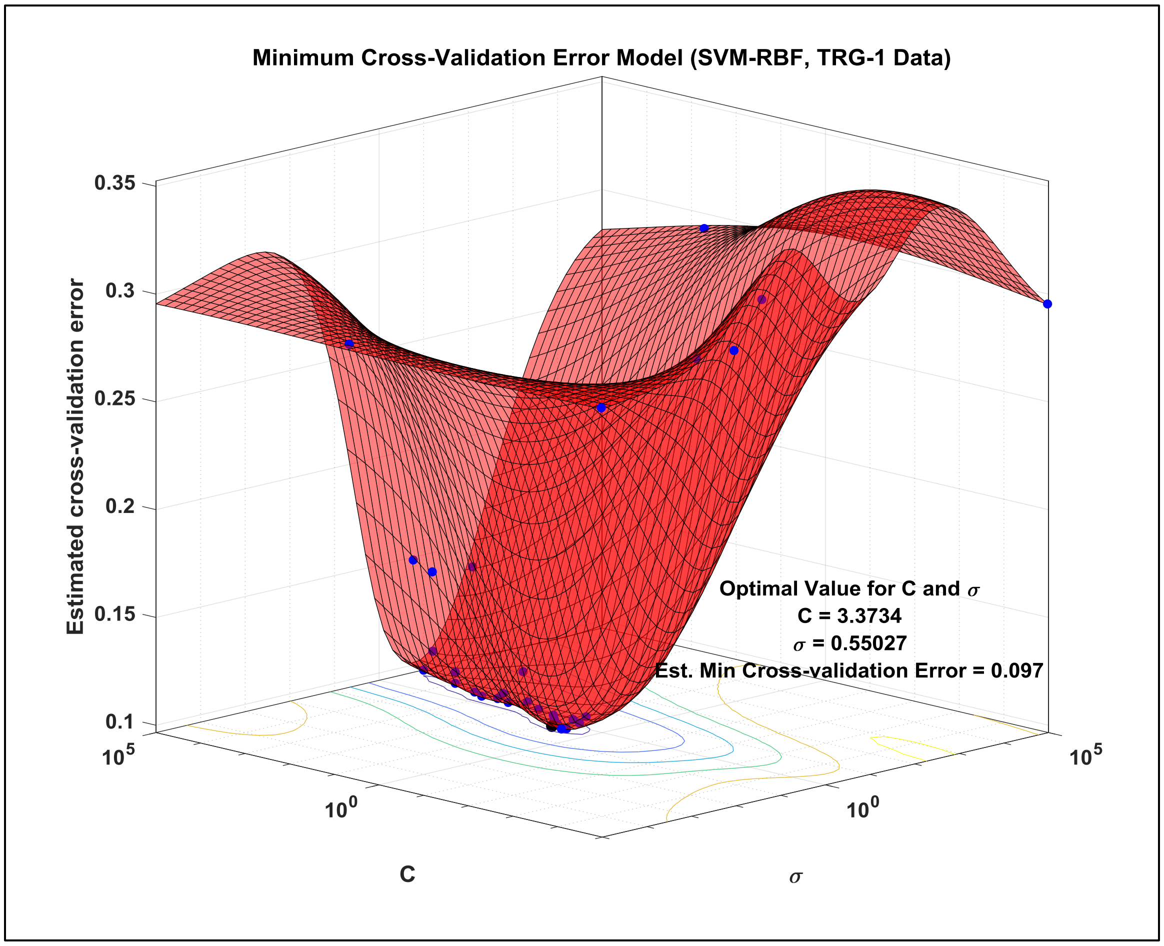


Figure 60: Bayes-optimized minimum CV-error; selection of C and σ (SVM-RBF, TRG-1 data).

In arriving at the optimal values for C and σ using Bayesian optimization, the number of objective function evaluations required to reach the minimum observed and estimated error is 30, as shown in Fig. 61. There was little to no improvement in the cross-validation error from function evaluation number 7 onwards.

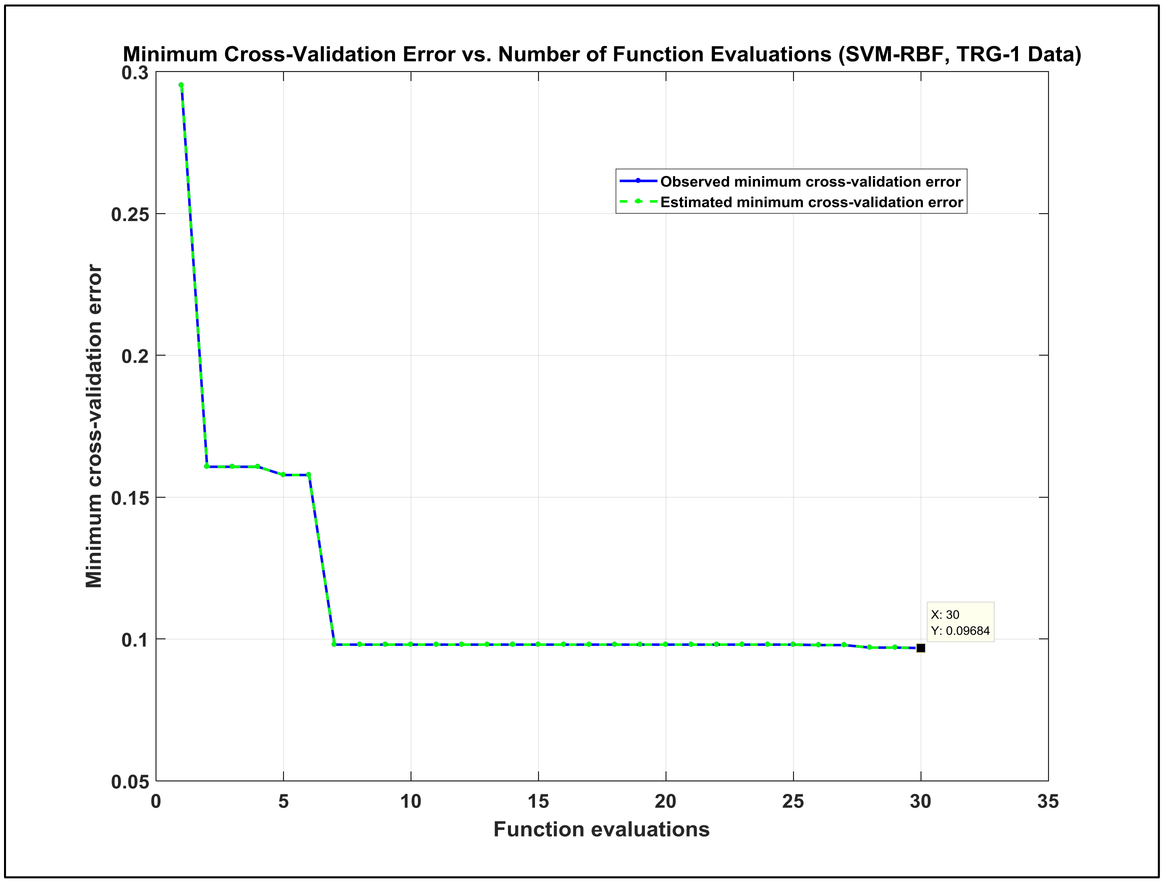


Figure 61: Minimum cross-validation error vs. number of function evaluations (SVM-RBF, TRG-1 data).

Similarly, the selection of SVM-CUBIC parameters was made using a Bayesian optimization method that minimized the 10-fold cross-validated loss, which for the SVM-CUBIC resulted in $C = 4917.8$ and $\sigma = 4.1674$ with a Bayes-estimated CV-error of 0.1011 (10.11%), as shown in Fig. 62.

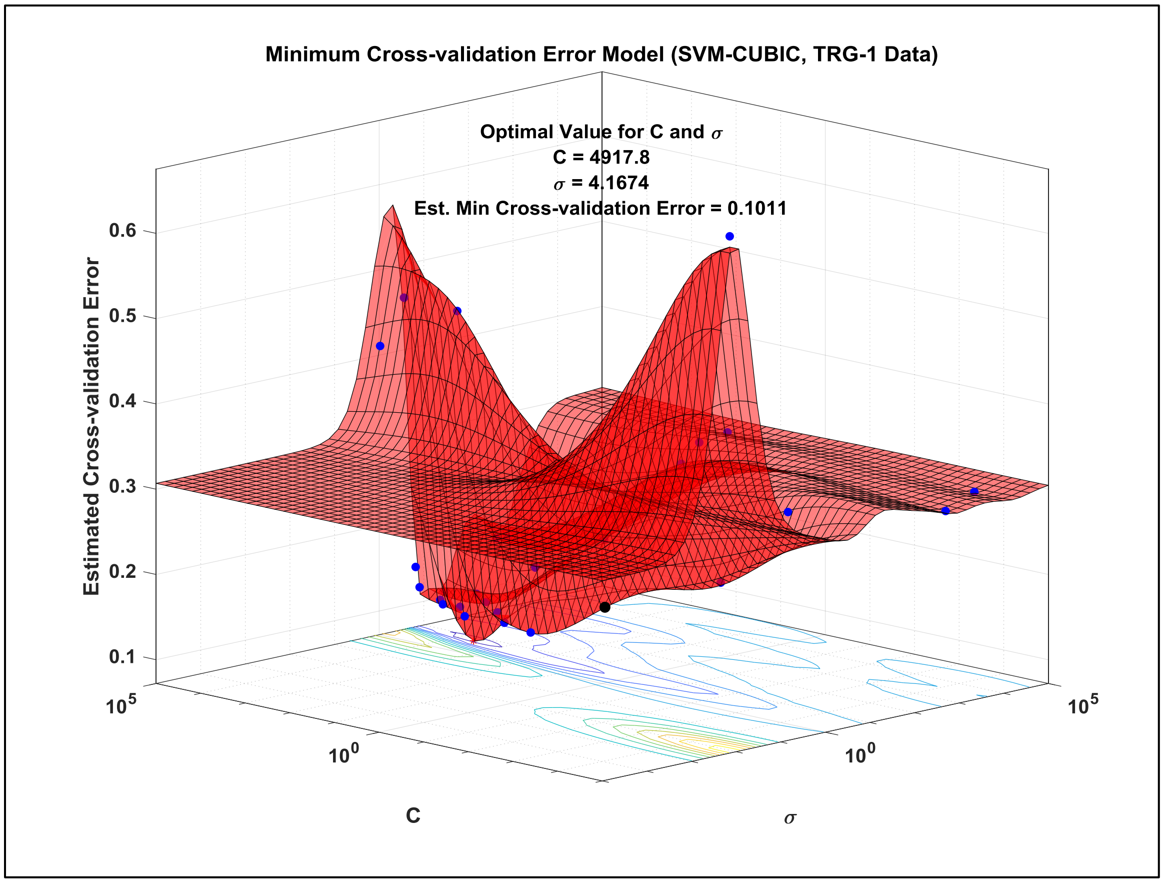


Figure 62: Bayes-optimized minimum CV-error; selection of C and σ (SVM-CUBIC, TRG-1 data).

The Bayes-estimated error for the SVM-CUBIC model 0.1011, which is in excellent agreement with the actual obtained error of 0.101. The Bayes optimization of values of C and σ that minimized the cross-validation error were determined to be $C = 4917.8$ and $\sigma = 4.1674$, while the number of function evaluations used in arriving at this result is shown in Fig. 63.

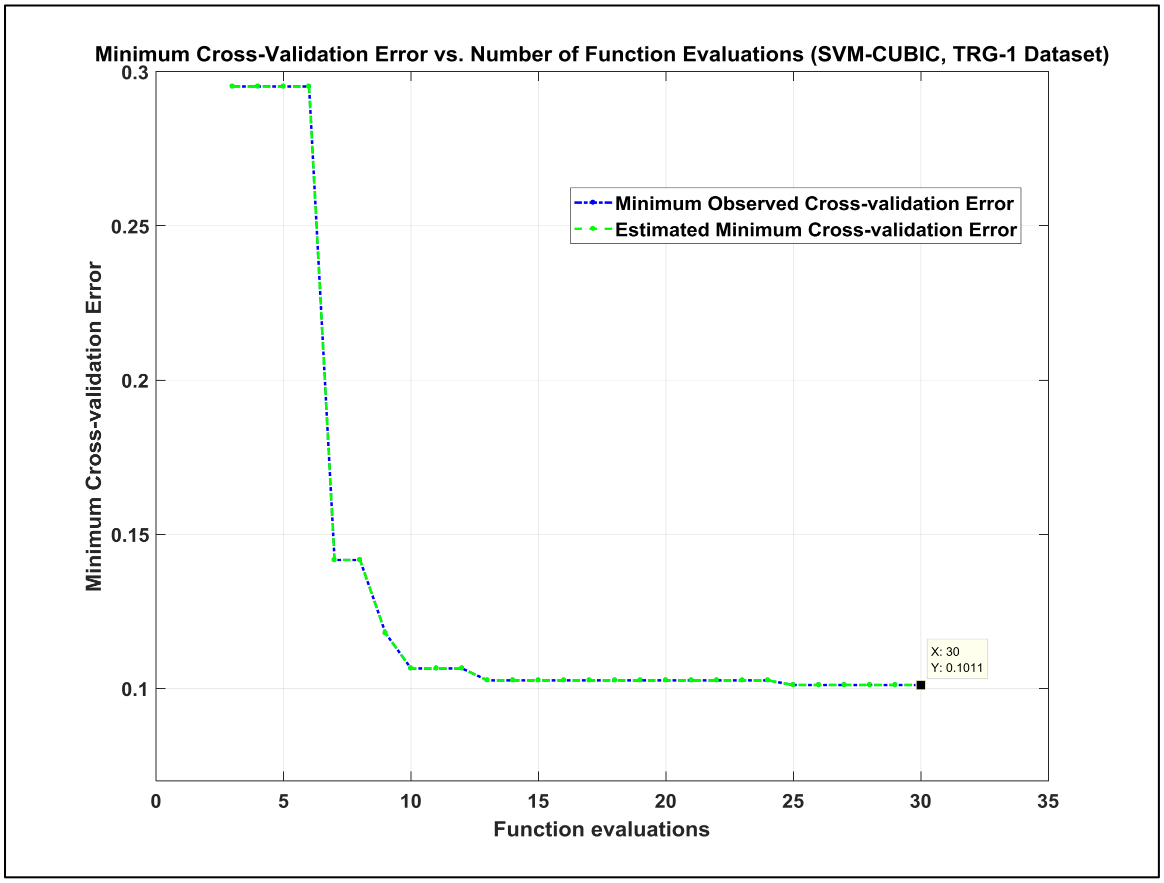


Fig. 63: Minimum cross-validation error vs. number of function evaluations (SVM-CUBIC, TRG-1 data).

The confusion matrix for the training scenario, that contains training performance for the SVM-RBF, SVM-CUBIC, ANN, and GPML-SE-ARD is shown in Fig. 64. As shown by the confusion matrix, the SVM-RBF classifier performed best with an overall training accuracy of 90.5%, followed by the ANN and GP-SE-ARD models performing almost equally well with an overall training accuracy of 90.2% and 90.1% respectively. The fourth-best classifier is the SVM-CUBIC model with an overall training accuracy of 89.9%. Even though the ANN accuracy during training was slightly lower than the accuracy of the SVM-RBF method, the model selection and training for the ANN was significantly faster.

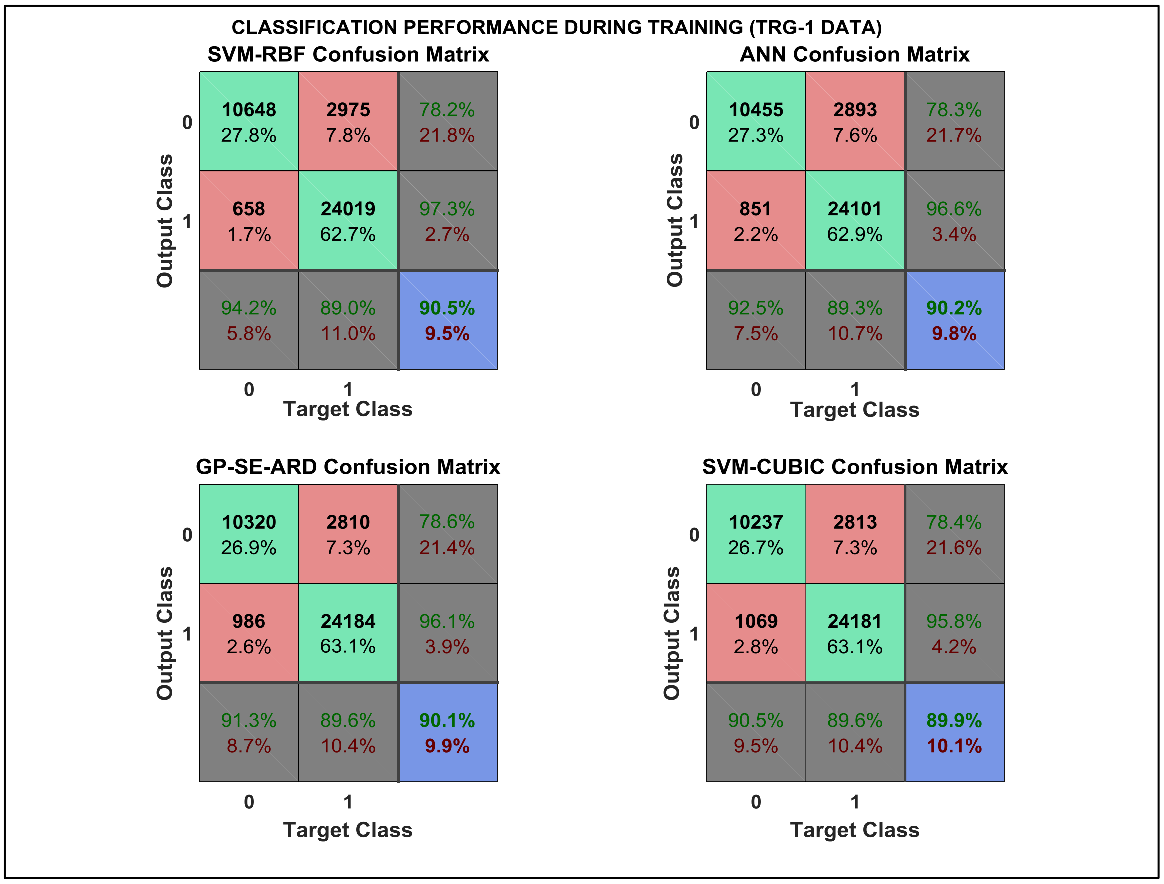


Figure 64: Classifier confusion matrix during training (TRG-1 data).

The second statistical classifier description, based on the confusion matrix, as introduced earlier in the chapter, is the ROC analysis. The ROC curve for all four classifiers is shown in Fig. 65. As shown in Fig. 65, all four classifiers are deemed excellent classifiers, based on the scale introduced earlier. The AUC for the SVM-RBF, ANN, GP-SE-ARD, and SVM-CUBIC models was determined to be 0.9546, 0.9531, 0.9577, and 0.9329, respectively. Following the training phase performance assessment, the test performance of the classifiers is assessed next. This is also referred to as the classifier generalization performance, or in other words, how well does the classifier generalize to data that it has not seen before.

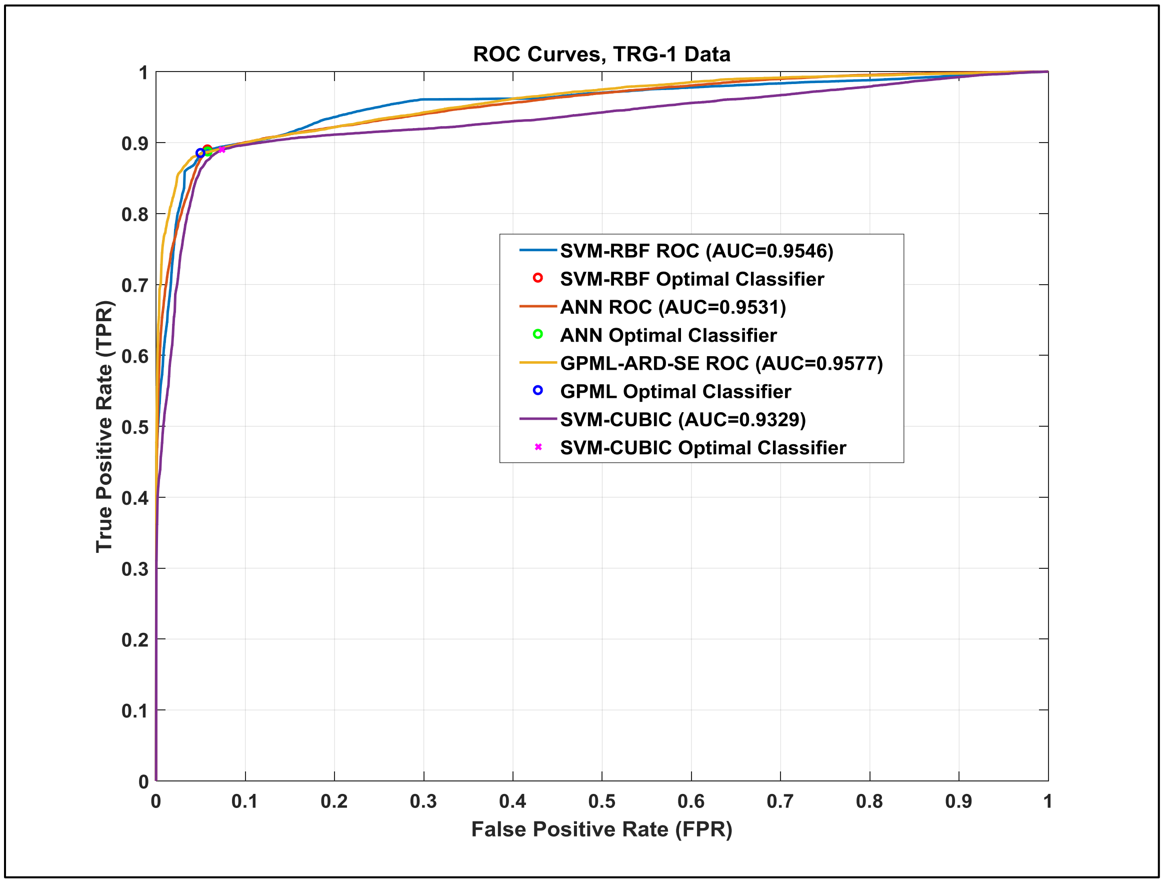


Figure 65: ROC analysis of top four performing classifiers (TRG-1 data).

Classifier Performance During Testing – TST-1 Data

To build the test performance matrix shown in Fig. 67, as well as the training performance matrix shown in Fig. 64, the probabilistic output of the ANN and the GP needs to be converted to a binary (upset/no upset) output. To do this, a decision boundary (threshold) needs to be determined above which the probability of upset is set to 1 and below which the probability of upset is set to 0 without compromising the accuracy of the machine. Fig. 66 shows a plot of prediction accuracy vs. threshold for TST-1 data and it indicates that reasonable threshold choice is 0.5, without compromising prediction accuracy.

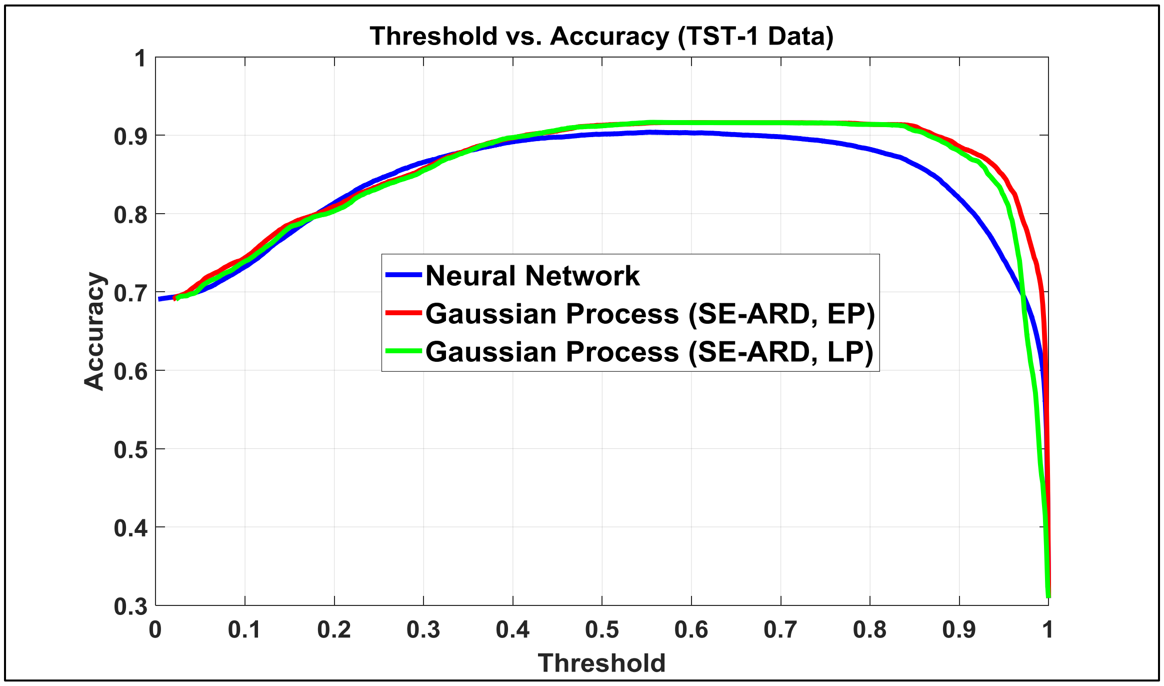


Figure 66: Threshold vs. accuracy for the TST-1 dataset.

The TST-1 test case contains 38,300 test points that will be used to assess the classifier generalization performance. As shown in Fig. 67, for the TST-1 test case, the SVM-RBF classifier performed best, with an overall accuracy of 91.5%, followed by the ANN, SVM-CUBIC, and GP-SE-ARD models with accuracies of 91%, 90.9%, and 90.7% respectively.

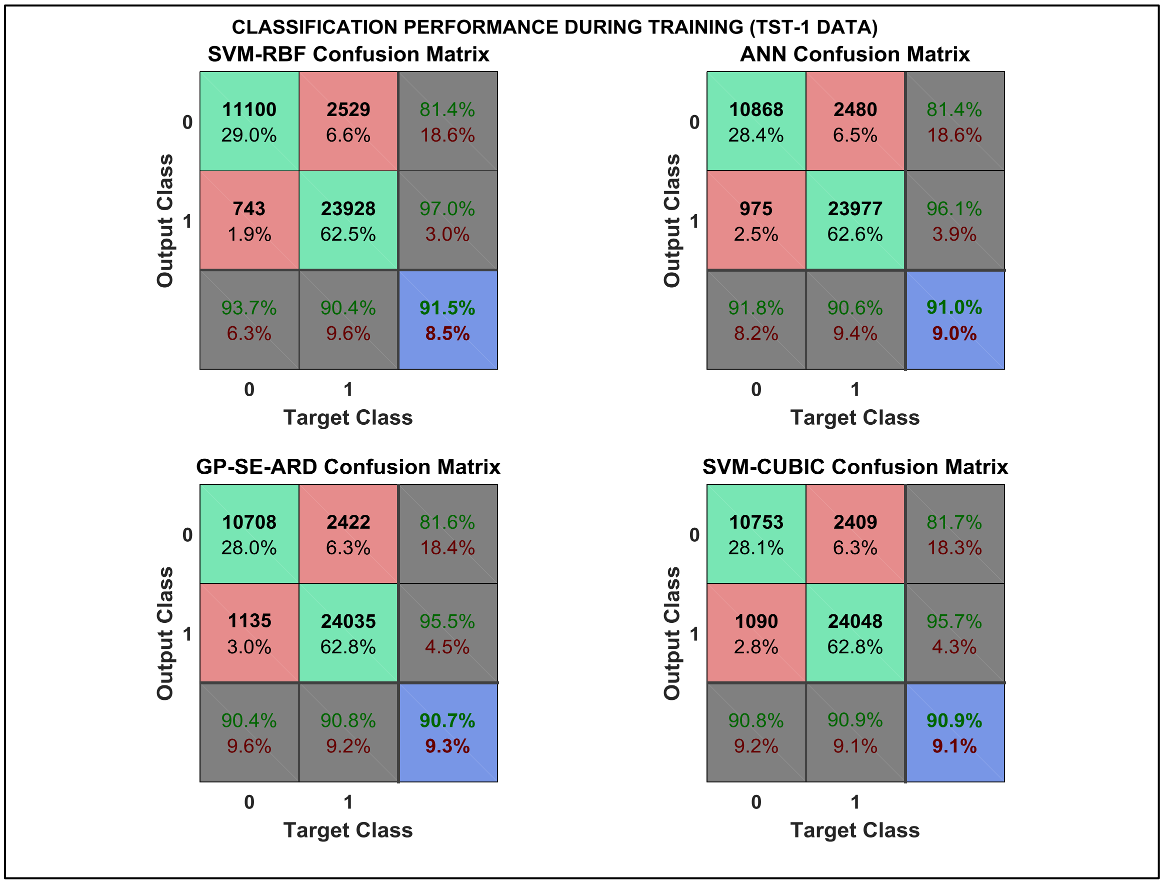


Figure 67: Classifier confusion matrix during testing (TST-1 data).

The ROC curve in Fig. 68 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.9469, 0.9390, 0.9528, and 0.9550 respectively, all of which are in excellent agreements with the respective training AUCs. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are “thresholded” at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset.

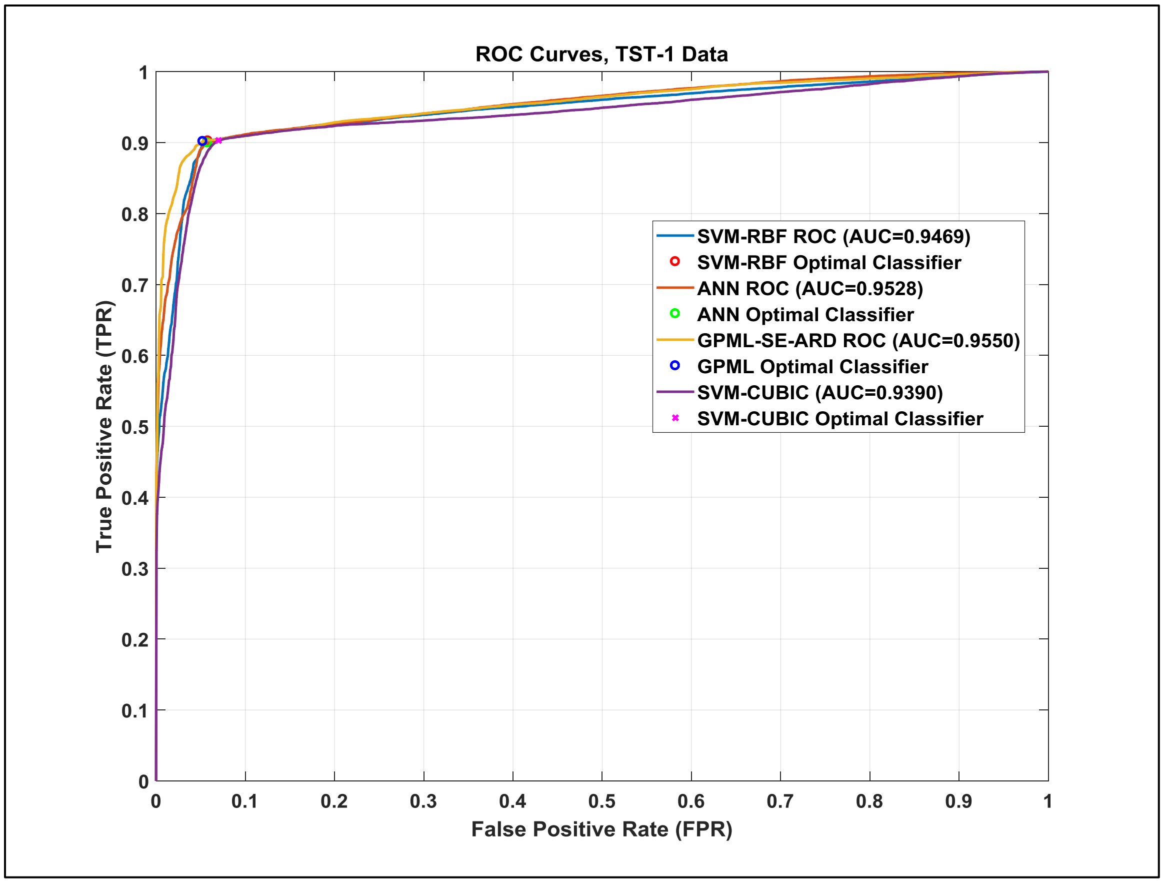


Figure 68: ROC analysis for top-four performing classifiers (TST-1 data).

The PoE plot illustrates how the excellent accuracy during training and generalization translates to the classifiers' ability to approximate the PoE as a function of RF pulse injection location for any given injected RF power level, frequency, and pulse width. The PoE plot in Fig. 69 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points, via the following formula:

$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}\!/_{\text{Number of Experiments}}$. The SVM-RBF classifier approximates the PoE curve for each clock edge extremely well. Most of the misclassification contained in the error rates stem from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. Fig. 69 shows how the excellent training and test performance translates to the approximation of the PoE plot as a function of injection location for a given injected RF power level, pulse width, and frequency.

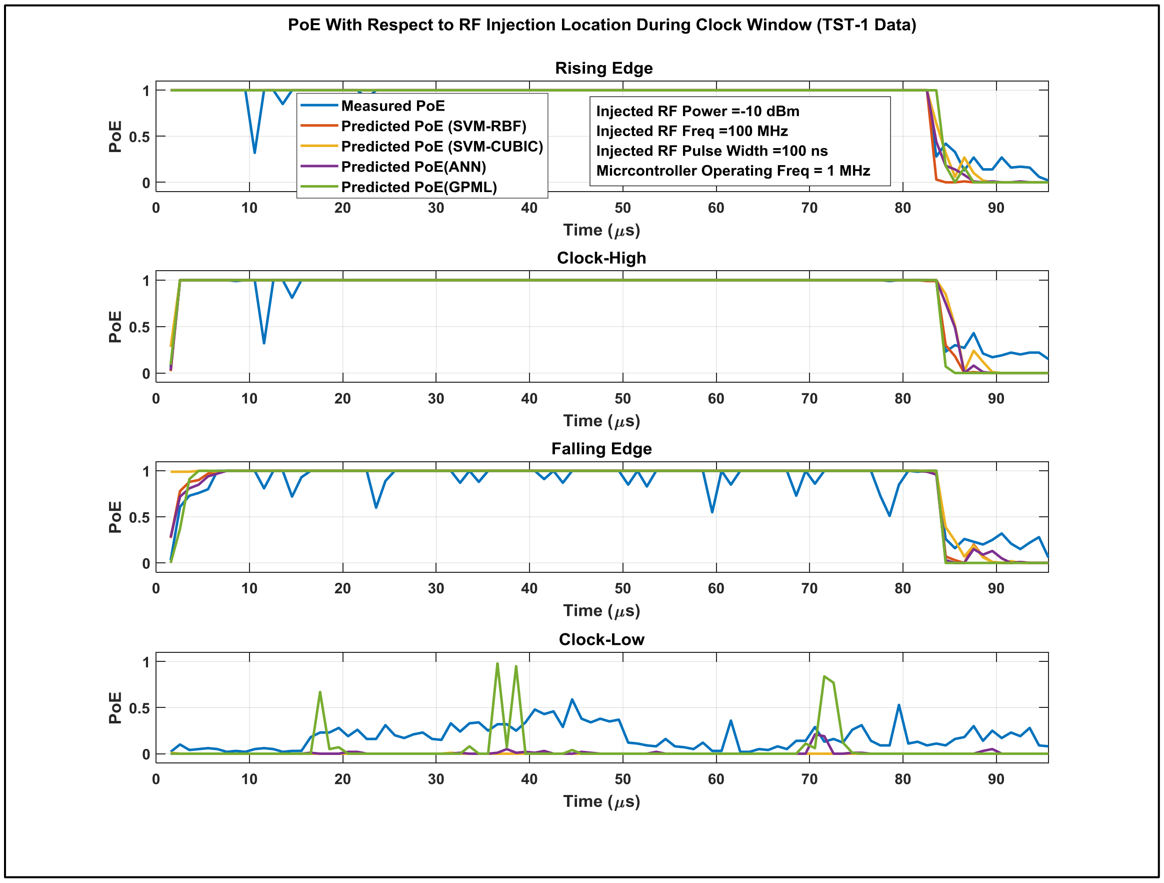


Figure 69: PoE vs. injection location for top four performing classifiers (TST-1 data).

As mentioned earlier, the PoE plot in Fig. 69 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points. All four classifiers approximate the PoE curves for each clock edge extremely well. As was the case with SVM-RBF-generated plots, most of the misclassification contained in the various error stems from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. The reason for the zero probability estimates of ANNs and GPMLs is that these two classifiers output a probability of an upset, which is then converted to a binary output of upset/no upset through a threshold with a decision boundary of 0.5. If the probability of upset is greater than 0.5, then it is converted to a 1 (upset), and if it is less than 0.5, it is converted to a 0 (no upset). A possible solution to the fair performance during clock low is to not threshold the probabilistic outputs of the GP and ANN, in other words, the solution could be to leave the ANN and GPML outputs in terms of their respective probability of upset. This solution will be explored in the discussion section that concludes this chapter.

To summarize results for the TE-1 test case, all three machine learning methods (SVM, ANN, and GPML) performed very well on TE-1 data, which consisted of 38,300 training points (TRG-1) and 38,300 test points (TST-1). The average training error across all three methods is 0.1102 (11.02%), meaning that they were almost 88.8% accurate, on average, with the actual accuracy ranging from 84.5% for the SVM-LIN classifier to 90.3% for the SVM-RBF classifier. Performance on the test data (TST-1) was also excellent. The average error across all classifiers for the test data is 10.2%, which is lower than the average training error of 11.02%. The best overall performing machine for TE-1 data is the support vector machine with a radial basis function kernel (SVM-RBF), even though the Bayes optimization of the model parameters and the training to determine the support vectors took the longest among the three methods. The execution time for training and test is not an objective of the research, and thus only a cursory qualitative assessment is provided. Classifier performance for the TE-1 data set was summarized in Table 12. The overall measured PoE for the TE-5 test case is 0.6908, with the machine-approximated values being 0.6442, 0.6500, and 0.6569 for the SVM-RBF, ANN, and GP-SE-ARD respectively.

Table 12: Classifier performance summary for the TE-1 data set.

TE-1 Data (TRG-1(N)=38,300 / TST-1(N)=38,300, TOTAL(N)=76,600) Injected RF Frequency $(f) = 100$ MHz Injected RF Power $(P) = -10$ dBm Injected RF Pulse Width $(PW) = 100$ ns		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel $C = 6.6988e-05, \quad \sigma = 4.3029e-04$ Bayes Optimization Est. Error: 0.1538	0.155	0.150
SVM – 2nd Degree Polynomial (Quadratic) Kernel $C = 3.0505, \quad \sigma = 0.7081$ Bayes Optimization Est. Error: 0.1072	0.107	0.096
SVM – 3rd Degree Polynomial (Cubic) Kernel $C = 4917.8, \quad \sigma = 4.1674$ Bayes Optimization Est. Error: 0.1011	0.101	0.091
SVM – Radial Basis Function (RBF/Gaussian) Kernel $C = 3.3734, \quad \sigma = 0.55027$ Bayes Optimization Est. Error: 0.09684	0.097	0.085
Artificial Neural Network (ANN)	0.100	0.090
Gaussian Processes for Machine Learning (GPML) Squared-Exponential Covariance Function w/ARD N=5000, T=38,300	0.101	0.100
Machine Learning Method	**Overall PoE (TST-1)**	
	Measured	**Predicted**
SVM - RBF	0.6908	0.6442
ANN		0.6500
GPML – SE-ARD		0.6569

Chapter 5-3 Test Experiment 2 (TE-2) Data Analysis

The data for test experiment 2 (TE-2) is divided into two sets. The first set of 38,300 data points is used to train SVM and ANN and select a model that provides the best fit, given the data (TRG-2 data). The GP will be trained on a randomly selected TRG-2 subset of 3,000 data points. The reason for the reduction in the training data set for the GP model from 5,000 to 3,000 was done to reduce training time. The quantitative analysis of training/prediction times is not within the scope of this research, as the primary assessment criteria for classifier performance is prediction accuracy. The second set within the TE-2 data set also consists of 38,300 test data points (TST-2 data). The TST-2 dataset will be used to test the machine and generate the TST-2 confusion matrix, ROC curves, and the PoE plot. None of the machines under consideration have been exposed to the TST-2 data set.

For the TE-2 data set, the injected RF frequency is $f = 100$ MHz, the injected RF power level is $P = -15$ dBm and the injected RF pulse width is $PW = 100$ ns. The RF pulse is injected into the clock pin, and it is timed to coincide with each clock edge (rising edge, clock high, falling edge, clock low) starting at time $t = 0.5\ \mu s$ until $t = 95\ \mu s$, which is the time window during which the MCU initializes and completes the counting operation. In sum, there are 383 injection locations during the clock window, each consisting of 100 shots (instance during which RF is injected into the clock pin), for a total of 38,300 data points. The first test run is used to collect 38,300 data points that will be used to train each machine, except the GP, which will be trained with a randomly selected subset of 3,000 data points. The test is repeated to collect an additional 38,300 data that will be used to test each machine.

The machines under consideration include SVMs with linear (SVM-LINEAR), quadratic (SVM-QUADRATIC), cubic (SVM-CUBIC), and Gaussian/radial basis function (SVM-RBF) kernels. The other two models used to model MCU upset include a two-layer feedforward neural network (ANN) that is trained using the scaled conjugate gradient and GPs with the squared exponential covariance functions with automatic relevance determination (GP-SE-ARD).

The performance of each machine is assessed during training and during the test phase. The training assessment features a Bayesian optimization analysis for the selection of key SVM parameters, a confusion matrix for training data, and ROC analysis for training data for all classifiers. The test assessment features a confusion matrix for test data, ROC analysis for test data, and a plot of the PoE as a function of injection location, injected RF power, injected RF frequency, and injected RF pulse duration for all classifiers.

Classifier Performance During Training – TRG-2 Data

In classification and regression scenarios, SVMs require that the kernel function (with its scale) be chosen along with the misclassification penalty parameter C. One of the most effective ways to do this is to utilize Bayesian optimization, as described in Section 3-5. The objective of Bayesian optimization here is to choose C and σ that will minimize the cross-validation error. Even though four different kernel types were analyzed for the SVM models, only the two best-performing SVM models will be presented in more detail for each test case, but the performance of all four kernel selections was summarized in Table 12.

For the training case (TRG-2) as well as the corresponding test case (TST-2), both of which contain 38,300 test points, the best performing SVM kernel is the SVM-RBF, while the second best SVM model is the SVM-CUBIC model. The selection of kernel parameters for all SVM kernels was made using a Bayesian optimization method that minimized the 10-fold cross-validation loss, which for the SVM-RBF resulted in $C = 80.649, and\ \sigma = 0.8429$ with a Bayes-estimated CV-error of 0.0272 (2.72%), as shown in Fig. 70.

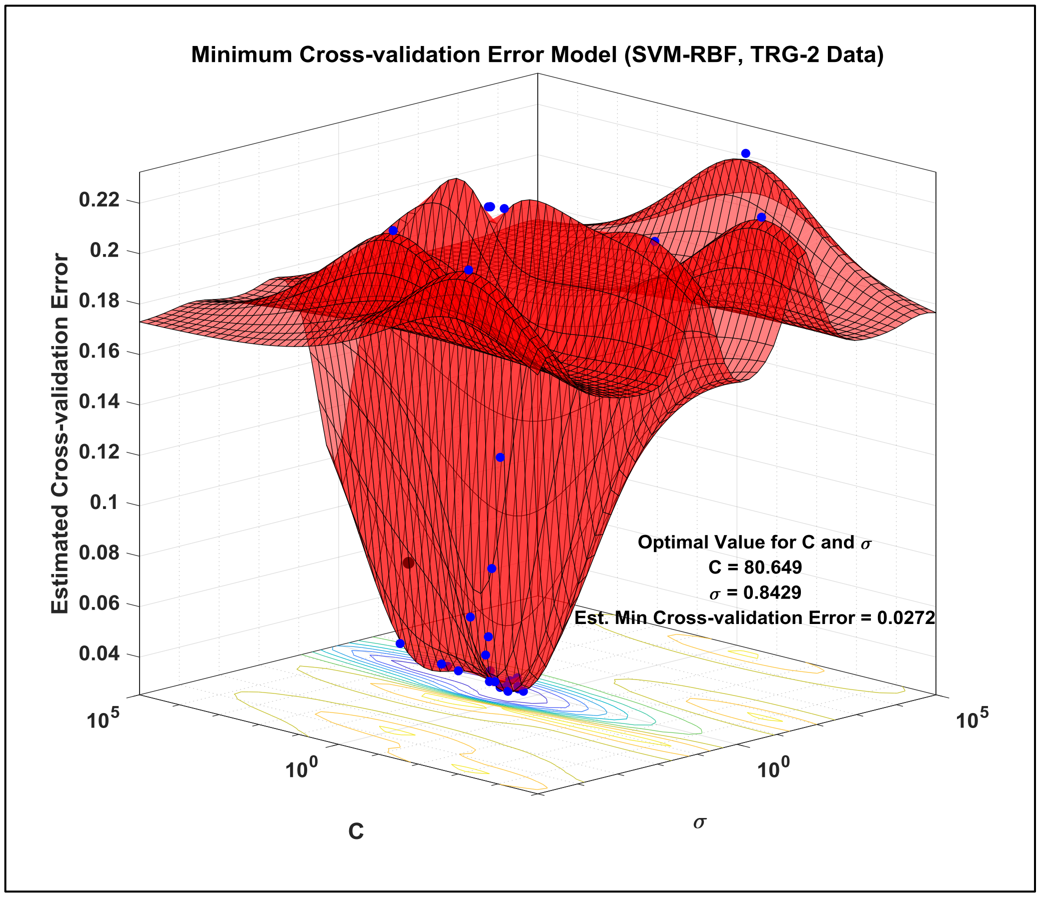


Figure 70: Bayes-optimized minimum CV-error; selection of C and σ (SVM-RBF, TRG-2 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations required to reach the minimum observed and estimated error is 30, as shown in Fig. 71**.**

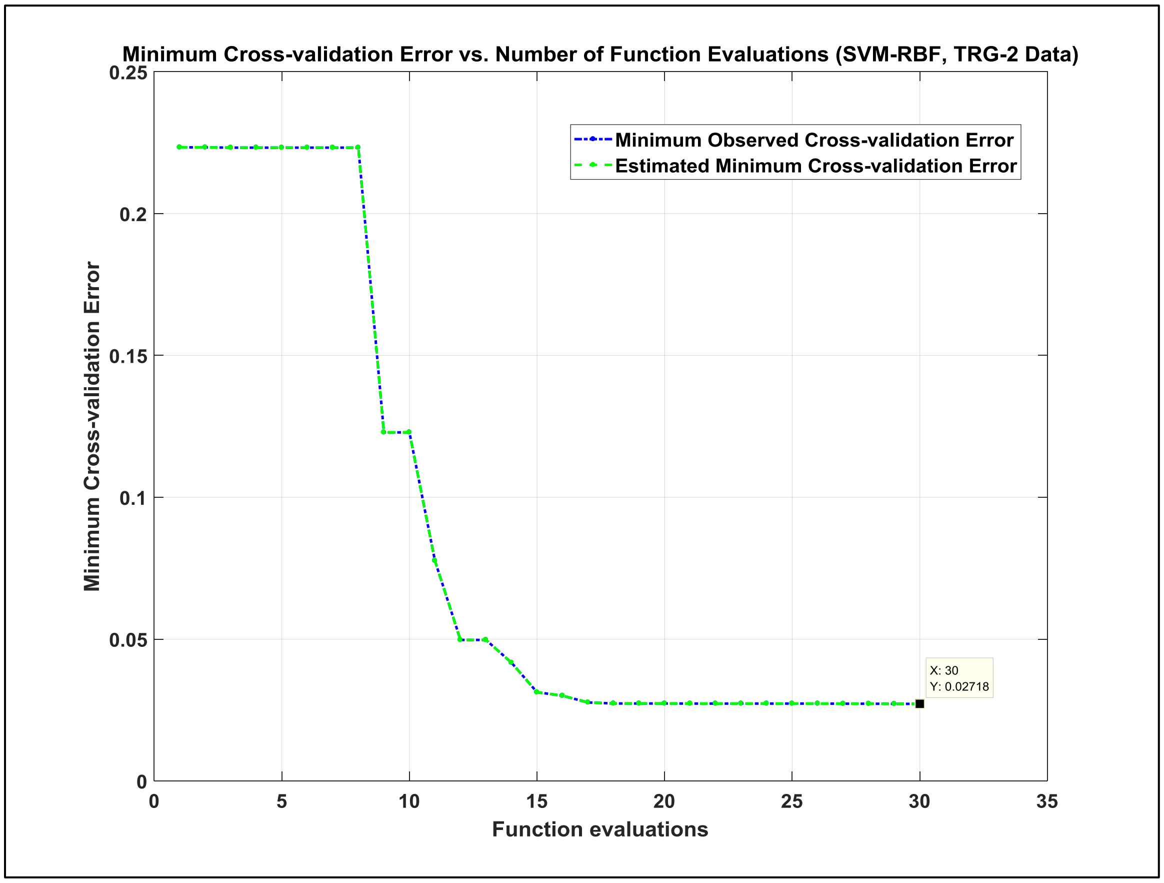


Figure 71: Cross-validation error vs. number of function evaluations (SVM-RBF, TRG-2 data).

The estimated as well as the observed cross-validation error of 0.0272 (2.72%) that were obtained using Bayesian optimization, were in excellent agreement with the actual SVM-RBF training cross-validation error of 0.0272 (2.72%).

The Bayes-estimated error for the SVM model with a cubic kernel machine is 0.0297, which is excellent agreement with the actual obtained error of 0.0298. The Bayes optimization of values of C and σ that minimized the cross-validation error were determined to be $C = 7.8099e - 05$ and $\sigma = 0.1535$, the model of which is shown in Fig. 72.

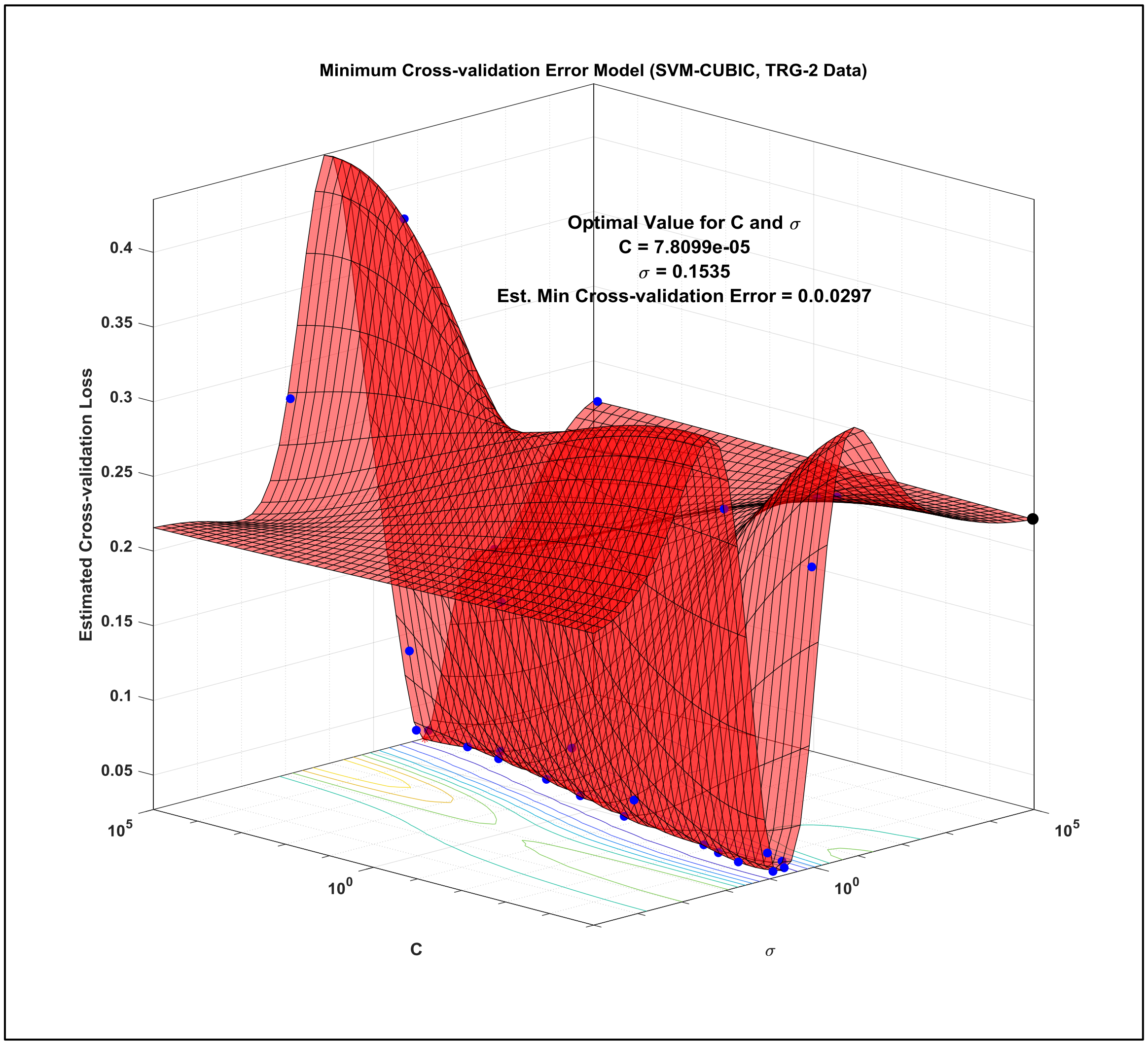


Figure 72: Bayes-optimized minimum CV-error; selection of C and σ (SVM-CUBIC, TRG-2 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations required to reach the minimum observed and estimated error is 30, as shown in Fig. 73**.**

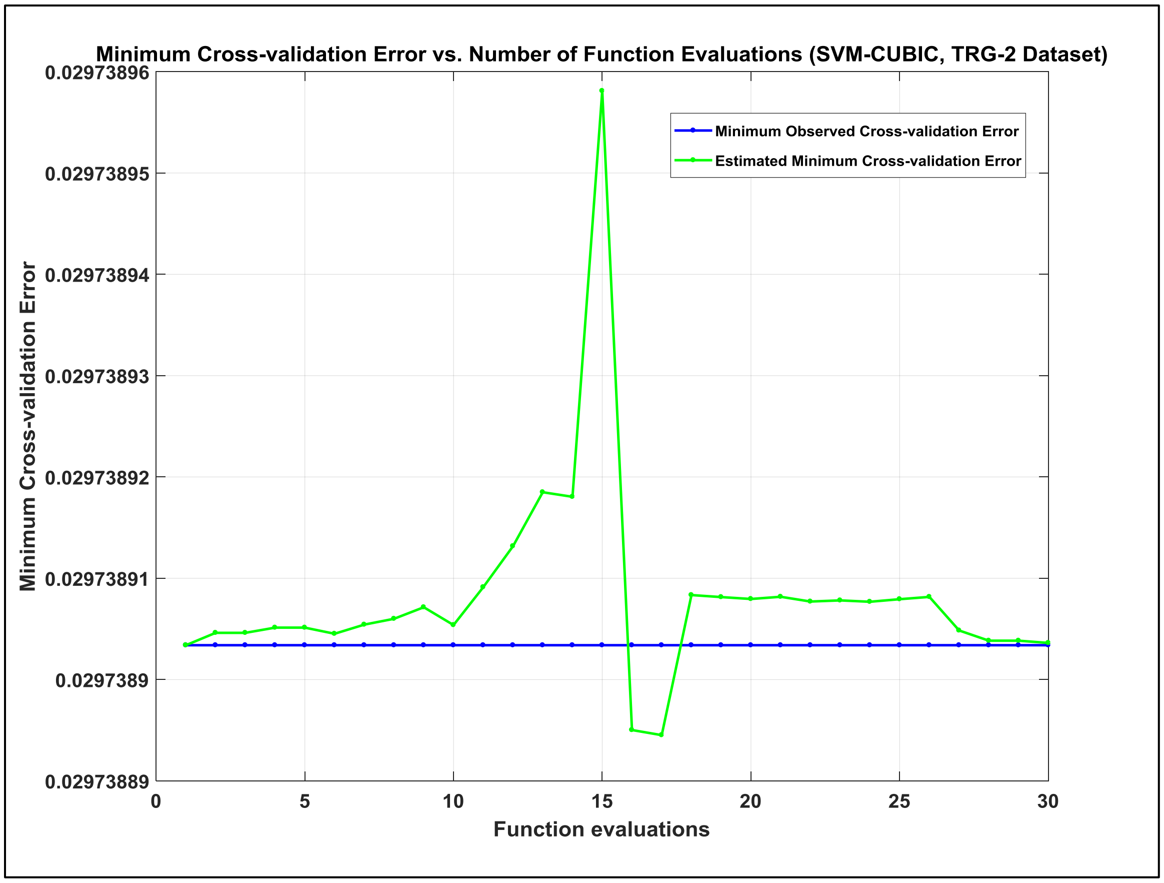


Figure 73: Cross-validation error vs. number of function evaluations (SVM-CUBIC, TRG-2 data).

The confusion matrix for the training scenario, that contains training performance for the SVM-RBF, SVM-CUBIC, ANN, and GPML-SE-ARD is shown in Fig. 74. As shown in Fig. 74, the SVM-RBF and the GP-SE-ARD classifier performed best with an overall accuracy of 97.3% and 97.2, respectively, followed by the ANN and SVM-CUBIC classifiers with accuracies of 97.1% and 96.6%, respectively.

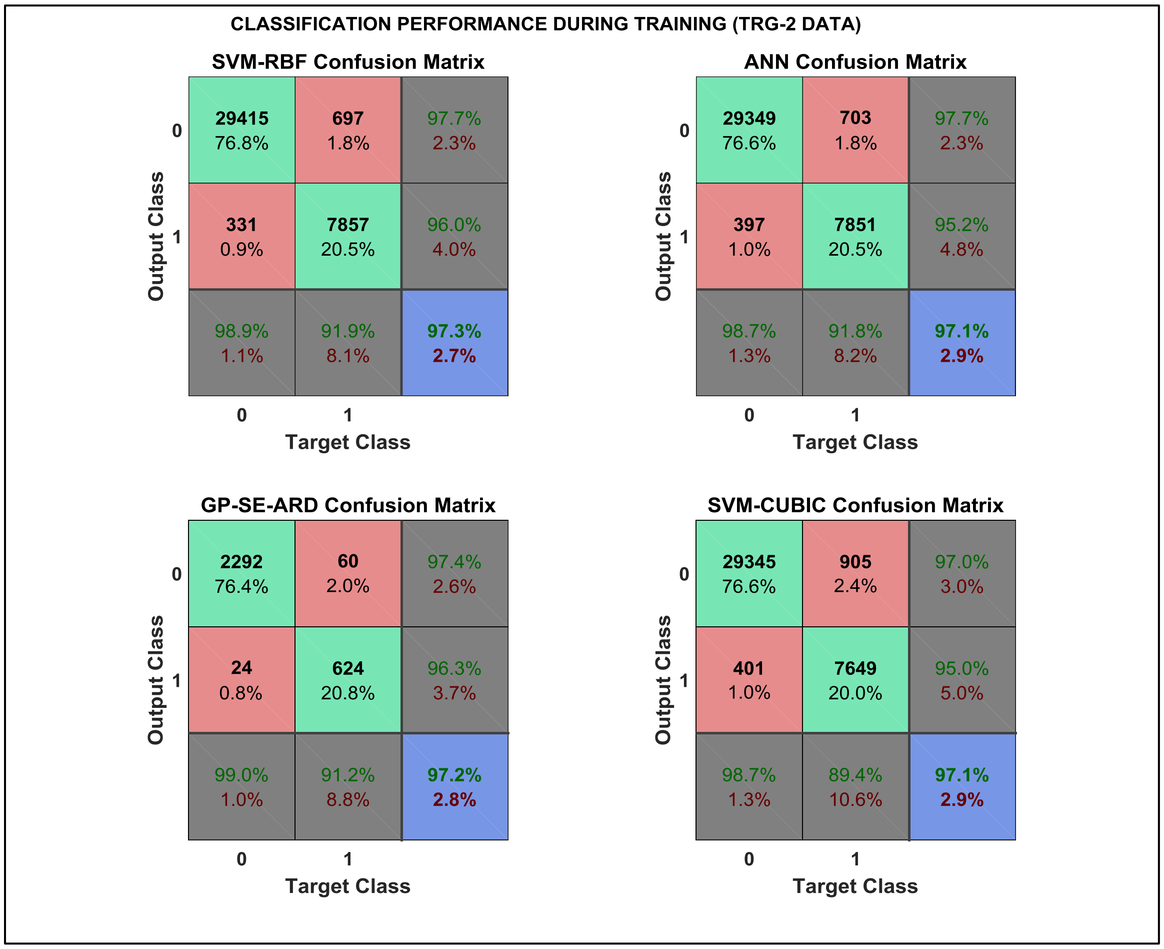


Figure 74: Classification performance during training (TRG-2 data).

The ROC curve in Fig. 75 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.9788, 0.9625, 0.9645, and 0.9790 respectively. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset.

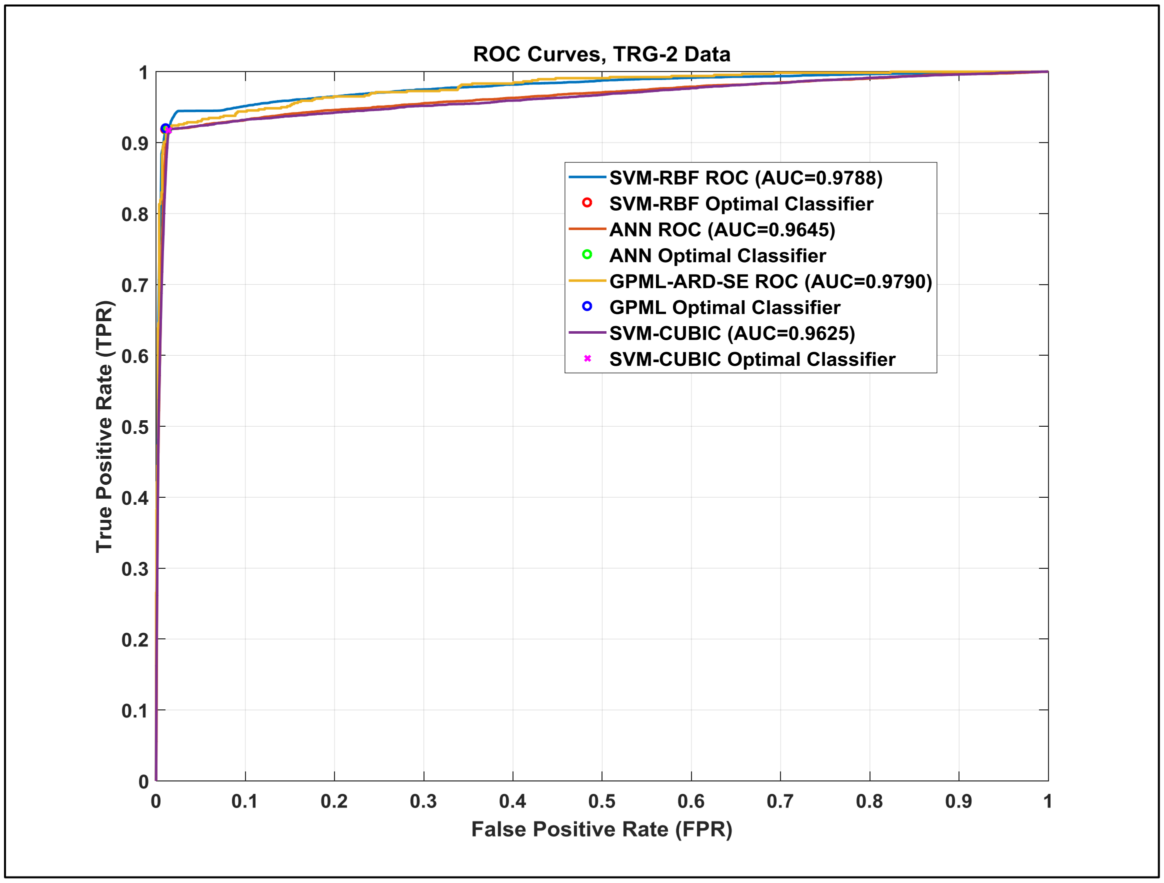


Figure 75: ROC analysis of top four performing classifiers (TRG-2 data).

As shown in Fig. 75, all four classifiers are excellent classifiers, based on the scale referenced earlier in the chapter. Following the training performance assessment, the test performance of the classifiers is assessed next. This is also referred to as the classifier generalization performance, or in other words, how well does the classifier generalize to data that it has not seen before.

Classifier Performance During Testing – TST-2 Data

To build the performance matrix shown in Fig. 77, the probabilistic output of the ANN and the GP needs to be converted to a binary (upset/no upset) output. To do this, a decision boundary (threshold) needs to be determined above which the probability of upset is set to 1 and below which the probability of upset is set to 0 without compromising the accuracy of the machine. Fig. 76 shows a plot of prediction accuracy vs. threshold for TST-2 data and it indicates that reasonable threshold choice is 0.5, without compromising prediction accuracy.

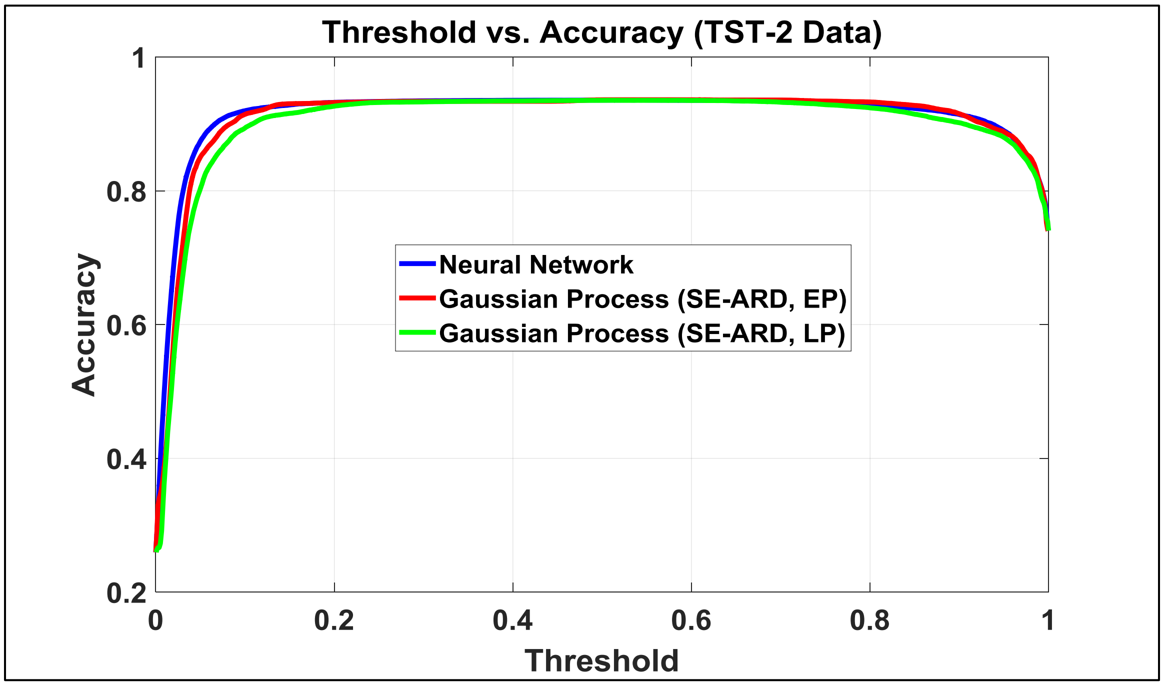


Figure 76: Threshold vs. accuracy for TST-2 dataset.

The TST-2 test case contains 38,300 test points that will be used to assess the classifier generalization performance. As shown in Fig. 77, on the classification for the TST-2 test case, the SVM-RBF, and the GP-SE-ARD classifier performed best, both of which have an overall accuracy of 93.6%, followed by the ANN and SVM-CUBIC classifiers with accuracies of 93.4% and 92.9%, respectively. The test performance of all classifiers was slightly worse than performance during training.

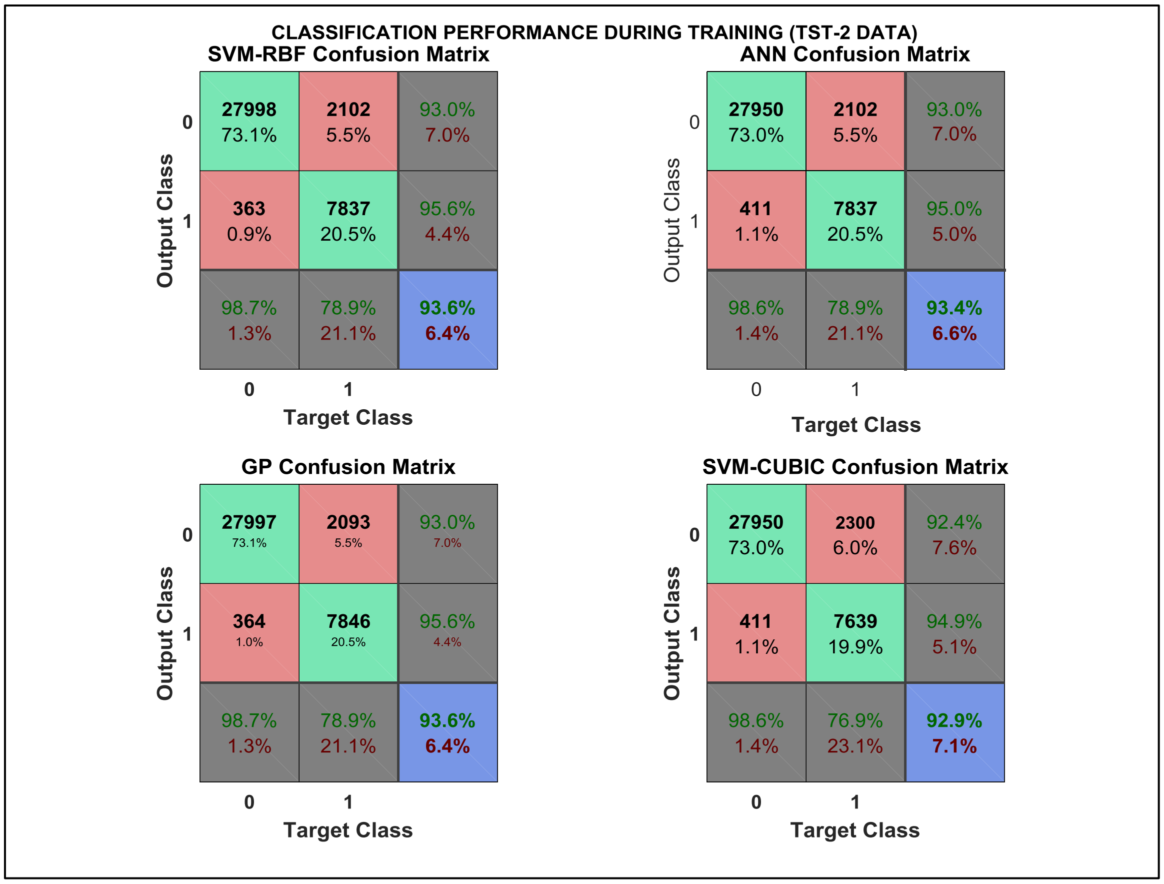


Figure 77: Classification performance during testing (TST-2 data).

The ROC curve in Fig. 78 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.8994, 0.8965, 0.9065, and 0.9001, all of which are in excellent agreements with the respective training AUCs. There is some miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are put through a threshold with a decision boundary at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset.

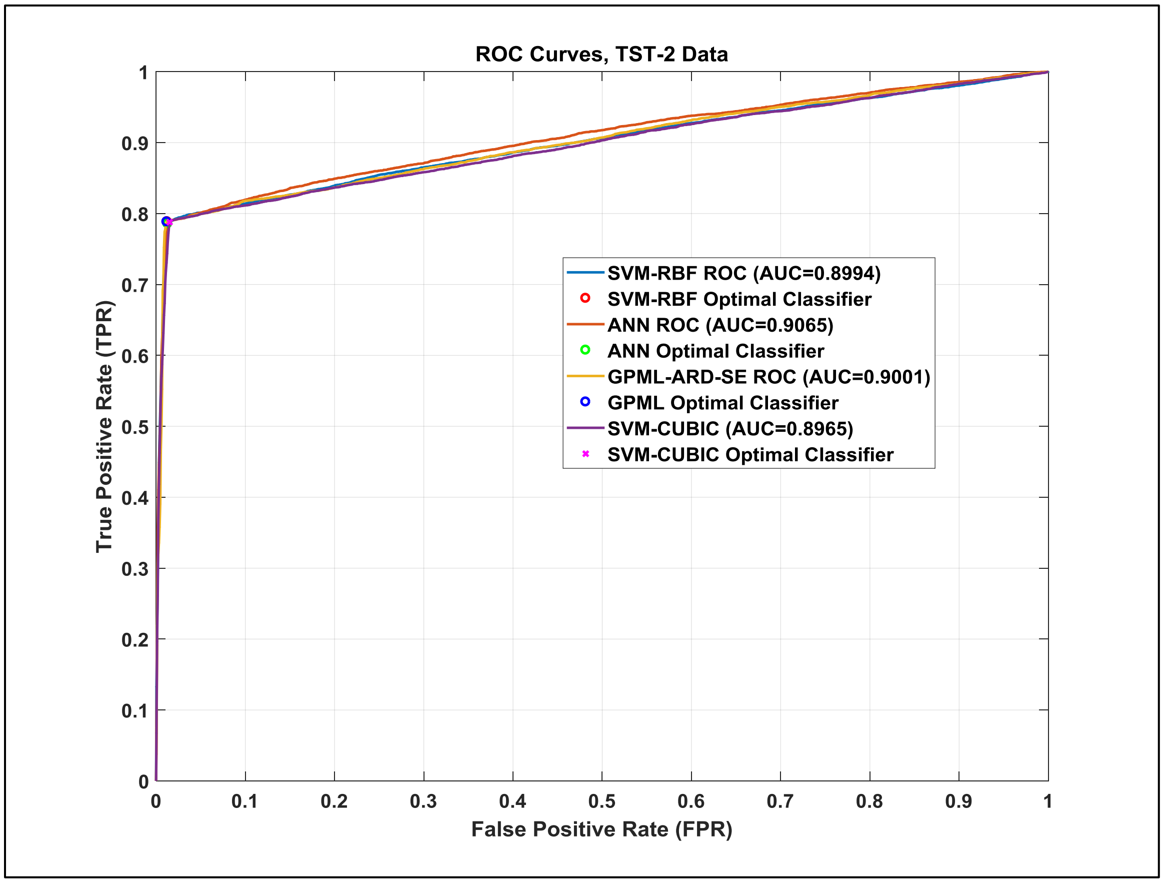


Figure 78: ROC analysis of top four performing classifiers during test (TST-2 data).

The PoE plot illustrates how the excellent accuracy during training and generalization translates to the classifiers' ability to approximate the PoE as a function of RF pulse injection location for any given injected RF power level, frequency, and pulse width. The PoE plot in Fig. 79 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points, via the following formula

$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}/_{\text{Number of Experiments}}$. The SVM-RBF classifier approximates the PoE curve for each clock edge extremely well. Most of the misclassification contained in the error rates stem from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that follows the analysis. Fig. 79 shows how the excellent training and test performance translates to the approximation of the PoE plot as a function of injection location for a given injected RF power level, pulse width, and frequency.

As mentioned earlier, the PoE plot in Fig. 79 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points. All four classifiers approximate the PoE curves for each clock edge extremely well. As was the case with SVM-RBF-generated plots, most of the misclassification contained in the various error stems from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. The reason for the zero probability estimates of ANNs and GPMLs is that these two classifiers output a probability of an upset, which is then converted to a binary output of upset/no upset through a threshold with a decision boundary of 0.5. If the probability of upset is greater than 0.5, then it is converted to a 1 (upset), and if it is less than 0.5, it is converted to a 0 (no upset). A possible solution to the fair performance during clock low is to not put the probability of upset data through the threshold mechanism, in other words, the solution could be to leave the ANN and GPML outputs in terms of their respective probability of upset. This solution will be explored in the discussion section that concludes this chapter.

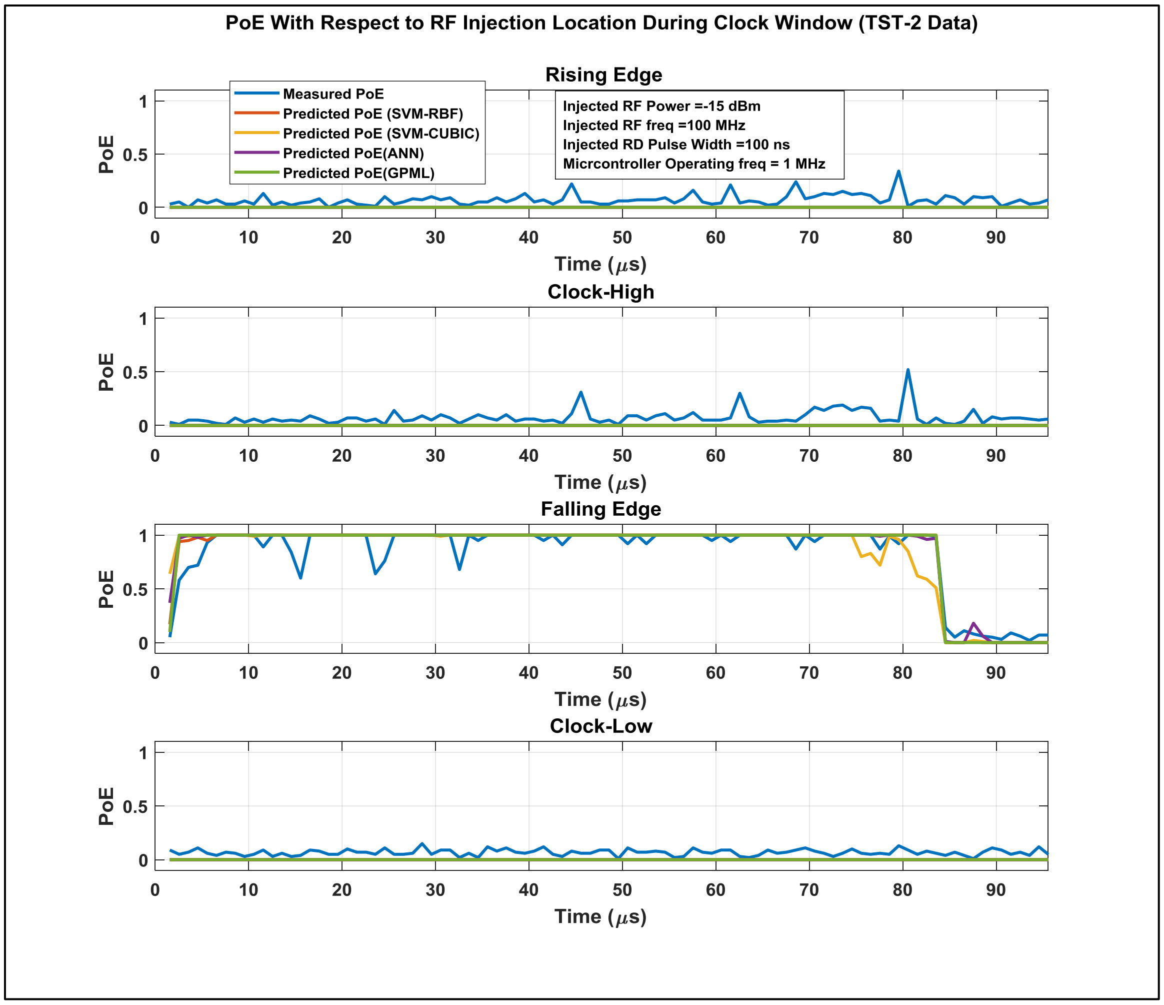


Figure 79: PoE vs. injection location for top four performing classifiers (TST-2 data set).

To summarize the TE-2 test case, all three machine learning methods (SVM, ANN, and GPML) performed very well on TE-2 data, which consisted of 38,300 training points (TRG-2) and 38,300 test points (TST-2). The average training error across all three methods is 0.0612 (6.12%), meaning that they were almost 93.88% accurate, on average, with the actual accuracy ranging from 77.67% for the SVM-LIN classifier to 97.28% for the SVM-RBF classifier. Performance on the test data (TST-2) was also excellent. The average error across all classifiers for the test data is 9.87%, which is higher than the average training error of 6.12%. The best overall performing machine for TE-2 data is the support vector machine with a radial basis function kernel (SVM-RBF), even though the Bayes optimization of the model parameters and the training to determine the support vectors took the longest to determine when compared to training times among the three methods. The execution time for training and test is not an objective of the research, and thus only a cursory qualitative assessment is provided. Classifier performance for the TE-2 dataset is summarized in Table 13. The overall measured PoE for the TE-2 test case is 0.2595, with the machine-approximated values being 0.2102, 0.2146, and 0.2144 for the SVM-RBF, ANN, and GP-SE-ARD, respectively.

Table 13: Classifier performance summary for the TE-2 data set.

TE-2 Data (TRG-2(N)=38,300 / TST-2(N)=38,300, TOTAL(N)=76,600) Injected RF Frequency $(f) = 100$ MHz Injected RF Power $(P) = -15$ dBm Injected RF Pulse Width $(PW) = 100$ ns		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel $C = 0.0807, \quad \sigma = 186.5021$ Bayes Optimization Est. Error = 0.2233	0.2233	0.2595
SVM – 2nd Degree Polynomial (Quadratic) Kernel $C = 0.5777, \quad \sigma = 0.3349$ Bayes Optimization Est. Error = 0.0309	0.0309	0.0681
SVM – 3rd Degree Polynomial (Cubic) Kernel $C = 7.8099e-05, \quad \sigma = 0.1535$ Bayes Optimization Est. Error = 0.0297	0.0298	0.071
SVM – Radial Basis Function (RBF/Gaussian) Kernel $C = 80.649, \quad \sigma = 0.8429$ Bayes Optimization Est. Error = 0.0272	0.0272	0.064
Artificial Neural Network (ANN)	0.0290	0.066
Gaussian Processes for Machine Learning (GPML) Squared-Exponential Covariance Function w/ARD N=3000, T=38,300	0.0270	0.064
Machine Learning Method	**Overall PoE (TST-2)**	
	Measured	**Predicted**
SVM – RBF	0.2595	0.2102
ANN		0.2146
GPML – SE-ARD		0.2144

Chapter 5-3 Test Experiment 3 (TE-3) Data Analysis

The data for test experiment 3 (TE-3) is divided into two sets. The first set of 38,300 data points is used to train SVM and ANN and select a model that provides the best fit, given the data (TRG-3 data). The GP will be trained on a randomly selected TRG-3 subset of 3,000 data points. The reason for the reduction in the training data set for the GP model from 5,000 to 3,000 was done to reduce training time. The quantitative analysis of training/prediction times is not within the scope of this research, as the primary assessment criteria for classifier performance is prediction accuracy. The second set within the TE-3 data set also consists of 38,300 test data points (TST-3 data). The TST-3 dataset will be used to test the machine and generate the TST-3 confusion matrix, ROC curves, and the PoE plot. None of the machines under consideration have been exposed to the TST-3 data set.

For the TE-3 data set, the injected RF frequency is $f = 100$ MHz, the injected RF power level is $P = -20$ dBm, and the injected RF pulse width is $PW = 100$ ns. The RF pulse is injected into the clock pin and it is timed to coincide with each clock edge (rising edge, clock-high, falling edge, clock-low) starting at time $t = 0.5\ \mu s$ until $t = 95\ \mu s$, which is the time window during which the MCU initializes and completes the counting operation. In sum, there are 383 injection locations during the clock window, each consisting of 100 shots (instance during which RF is injected into the clock pin), for a total of 38,300 data points. The first test run is used to collect 38,300 data points that will be used to train each machine, except the GP, which will be trained with a randomly selected subset of 3,000 data points. The test is repeated to collect an additional 38,300 data that will be used to test each trained machine.

The machines under consideration include SVMs with linear (SVM-LINEAR), quadratic (SVM-QUADRATIC), cubic (SVM-CUBIC), and Gaussian/radial basis function (SVM-RBF) kernels. The other two models used to model MCU upset include a two-layer feedforward neural network (ANN) that is trained using the scaled conjugate gradient and GPs with the squared exponential covariance functions with automatic relevance determination (GP-SE-ARD).

The performance of each machine is assessed during training and during the test phase. The training assessment features a Bayesian optimization analysis for the selection of key SVM parameters, a confusion matrix for training data, and ROC analysis for training data for all classifiers. The test assessment features a confusion matrix for test data, ROC analysis for test data, and a plot of the PoE as a function of injection location, injected RF power, injected RF frequency, and injected RF pulse duration for all classifiers.

Classifier Performance During Training – TRG-3 Data

In classification and regression scenarios, SVMs require that the kernel function (with its scale) be chosen along with the misclassification penalty parameter C. One of the most effective ways to do this is to utilize Bayesian optimization, as described in Section 3-5. The objective of Bayesian optimization here is to choose C and σ that will minimize the cross-validation error. Even though four different kernel types were analyzed for the SVM models, only the two best-performing SVM models will be presented in more detail for each test case, but the performance of all four kernels selections is summarized in Table 14 at the end of the section.

For the training case (TRG-3) as well as the corresponding test case (TST-3), both of which contain 38,300 test points, the best performing SVM kernel is the RBF/Gaussian kernel (SVM-RBF), while the second best SVM kernel is the polynomial kernel of third degree (SVM-CUBIC). The selection of kernel parameters for all SVM kernels was made using a Bayesian optimization method for which the objective function was the minimum 10-fold cross-validation loss, which for the SVM-RBF resulted in $C = 7.4541$ and $\sigma = 0.9517$ with a Bayes-estimated CV-error of 0.0192 (1.92%), as shown in Fig. 80**.**

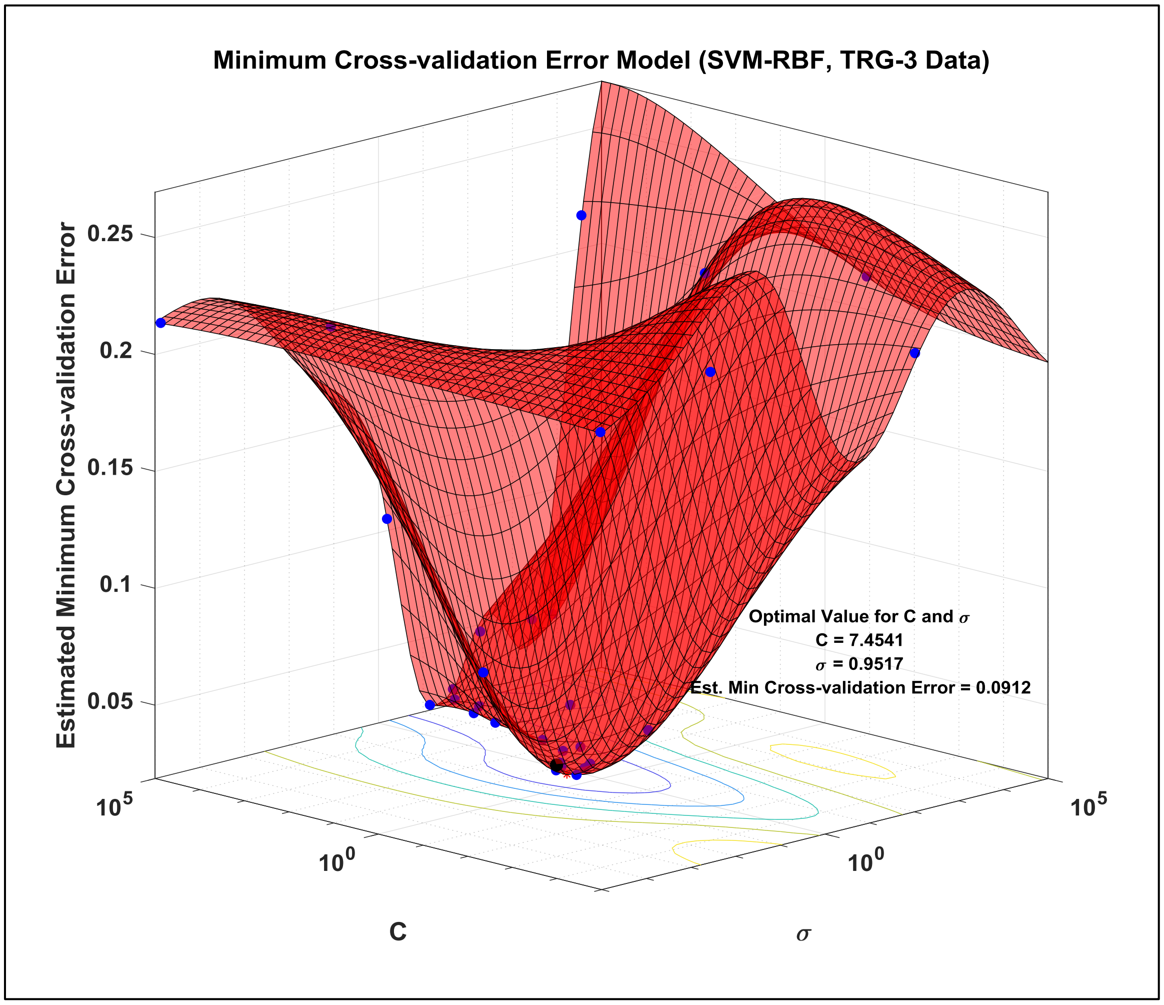


Figure 80: Bayes-optimized minimum CV-error; selection of C and σ (SVM-RBF, TRG-3 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 81**.**

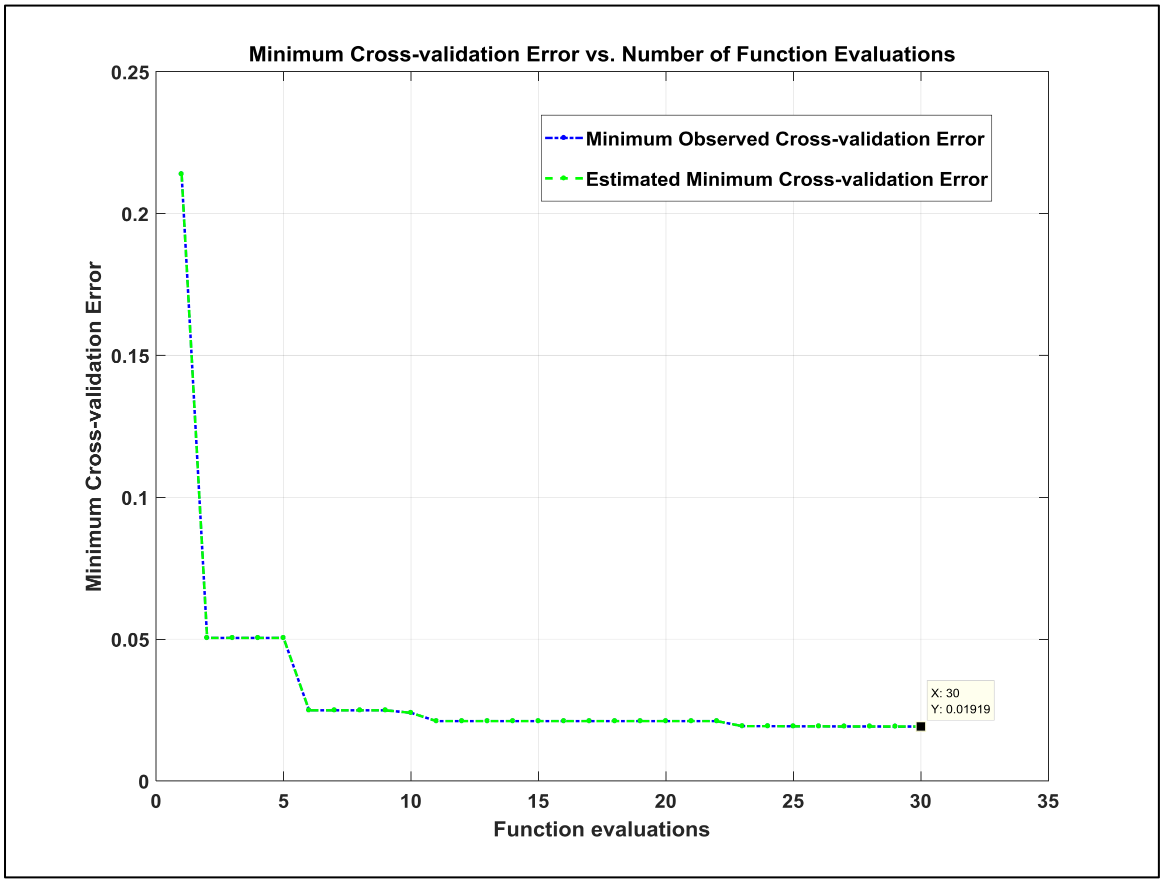


Figure 81: Cross-validation error vs. number of function evaluations (SVM-RBF, TRG-3 data).

The estimated, as well as the observed cross-validation errors of 0.0192% (1.92%) for the SVM-RBF model, obtained using Bayesian optimization, were in excellent agreement with the actual SVM-RBF training cross-validation error of 0.0192 (1.92%).

The Bayes-estimated error for the SVM model with a cubic kernel is 0.0196, which is excellent agreement with the actual obtained error of 0.0200. The Bayes optimization of values of C and σ that minimized the cross-validation error for the SVM-

CUBIC model were determined to be $C = 1.2546e - 04$ and $\sigma = 486.4421$, which is shown in Fig. 82.

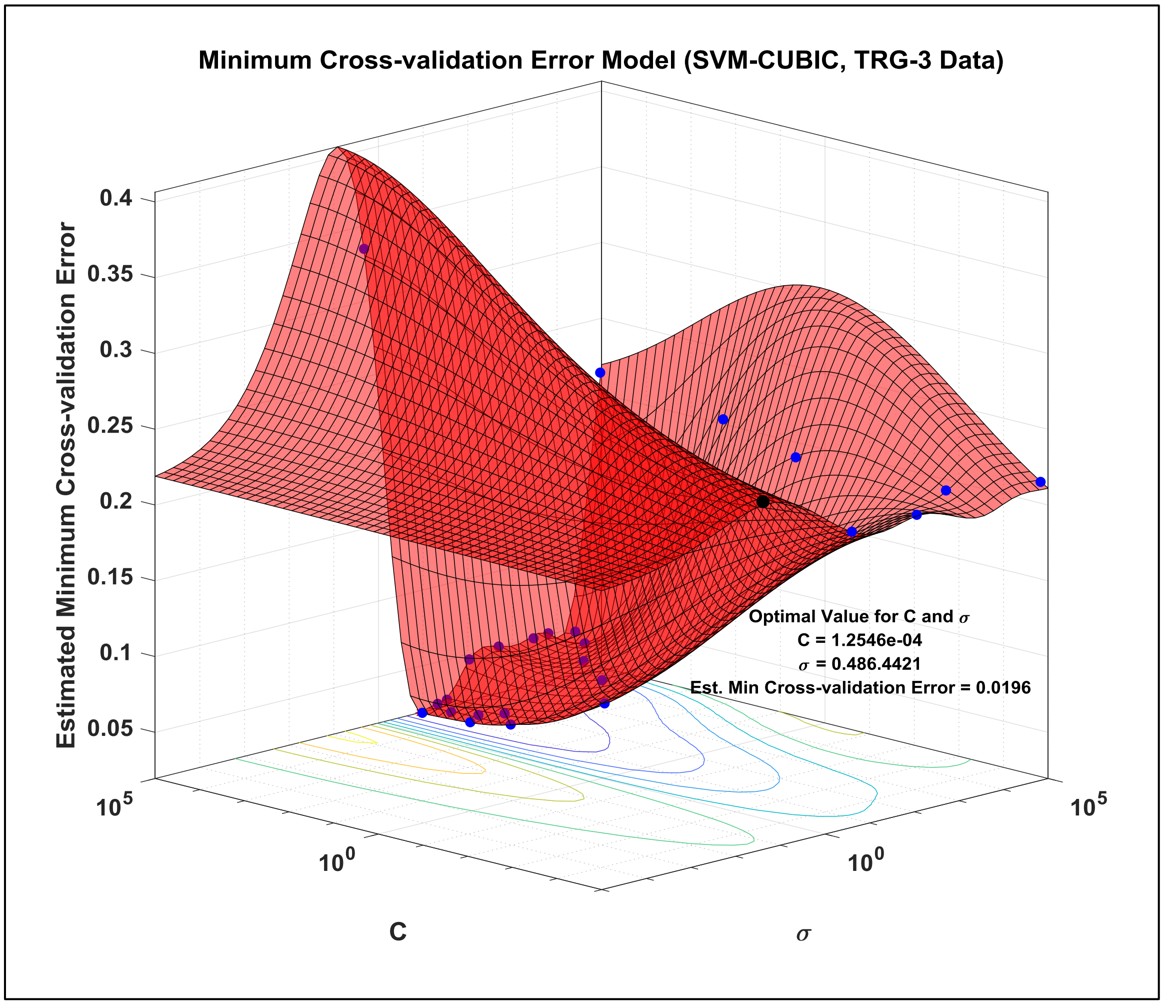


Figure 82: Bayes-optimized minimum CV-error; selection of C and σ (SVM-CUBIC, TRG-3 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 83**.**

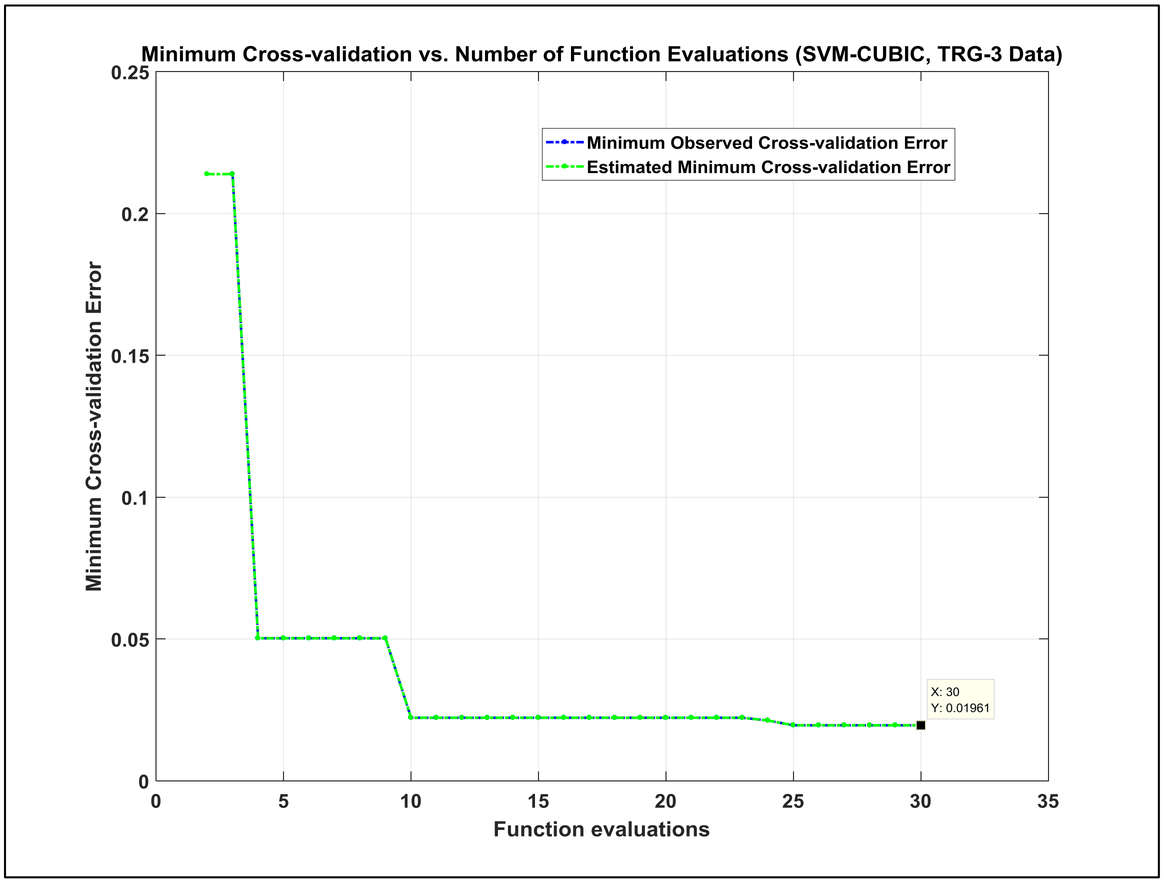


Figure 83: Cross-validation error vs. number of function evaluations (SVM-CUBIC, TRG-3 data).

The estimated, as well as the observed cross-validation error of 0.01961% (1.961%) for the SVM-CUBIC model that were obtained using Bayesian optimization, were in excellent agreement with the actual SVM-CUBIC training cross-validation error of 0.0200 (2.00%).

The confusion matrix for the training scenario (TRG-3), that contains training performance for the SVM-RBF, SVM-CUBIC, ANN, and GPML-SE-ARD is shown in Fig. 84.

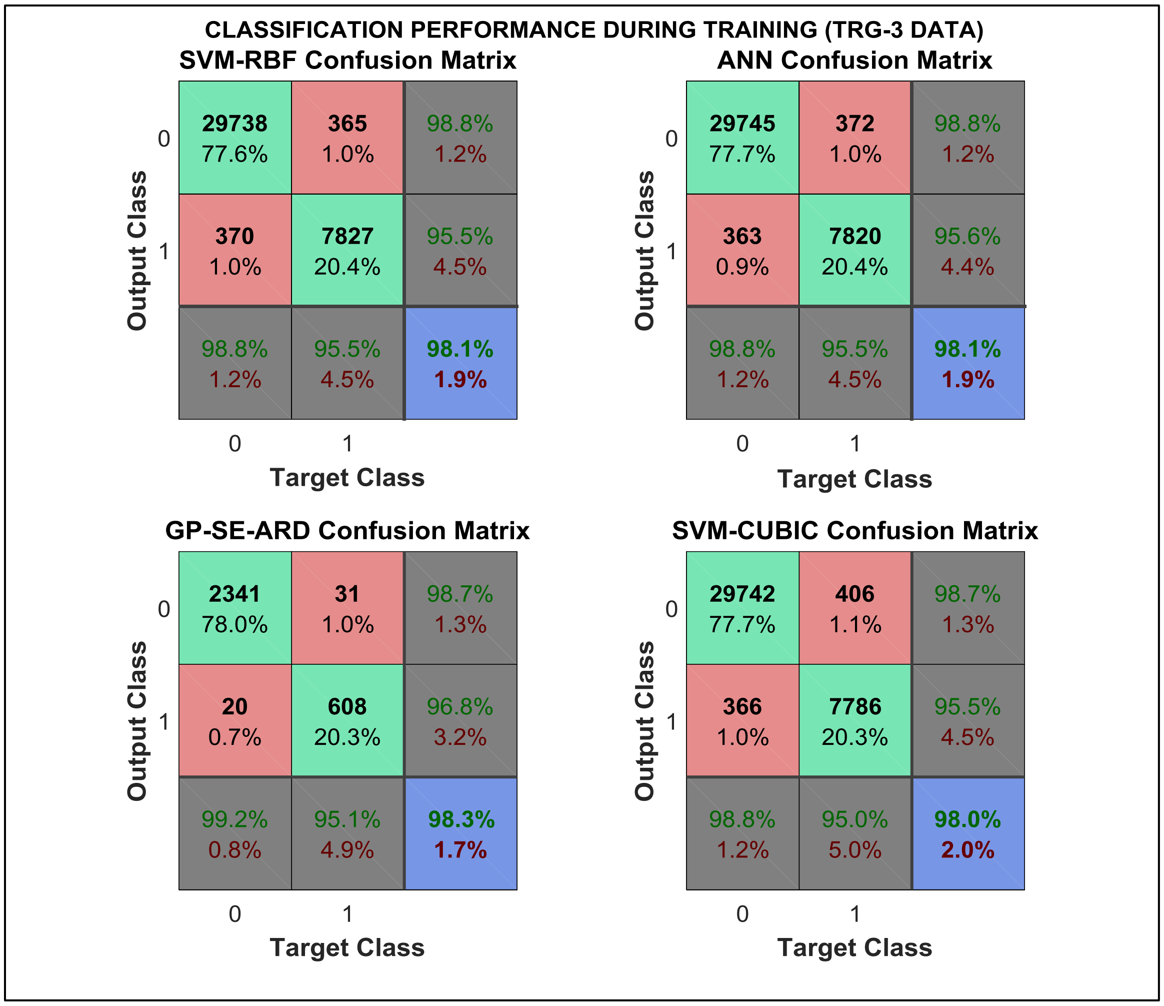


Figure 84: Classification training performance for TRG-3 data.

As shown by the confusion matrix, the GP-SE-ARD classifier performed best with an overall training accuracy of 98.3%, with the ANN and SVM-RBF models performing equally well with an overall training accuracy of 98.1%. The fourth-best classifier is an SVM with a cubic kernel and an overall training accuracy of 98%.

The second statistical classifier description, based on the confusion matrix, is the ROC analysis. The ROC curve for all four classifiers is shown in Fig. 85.

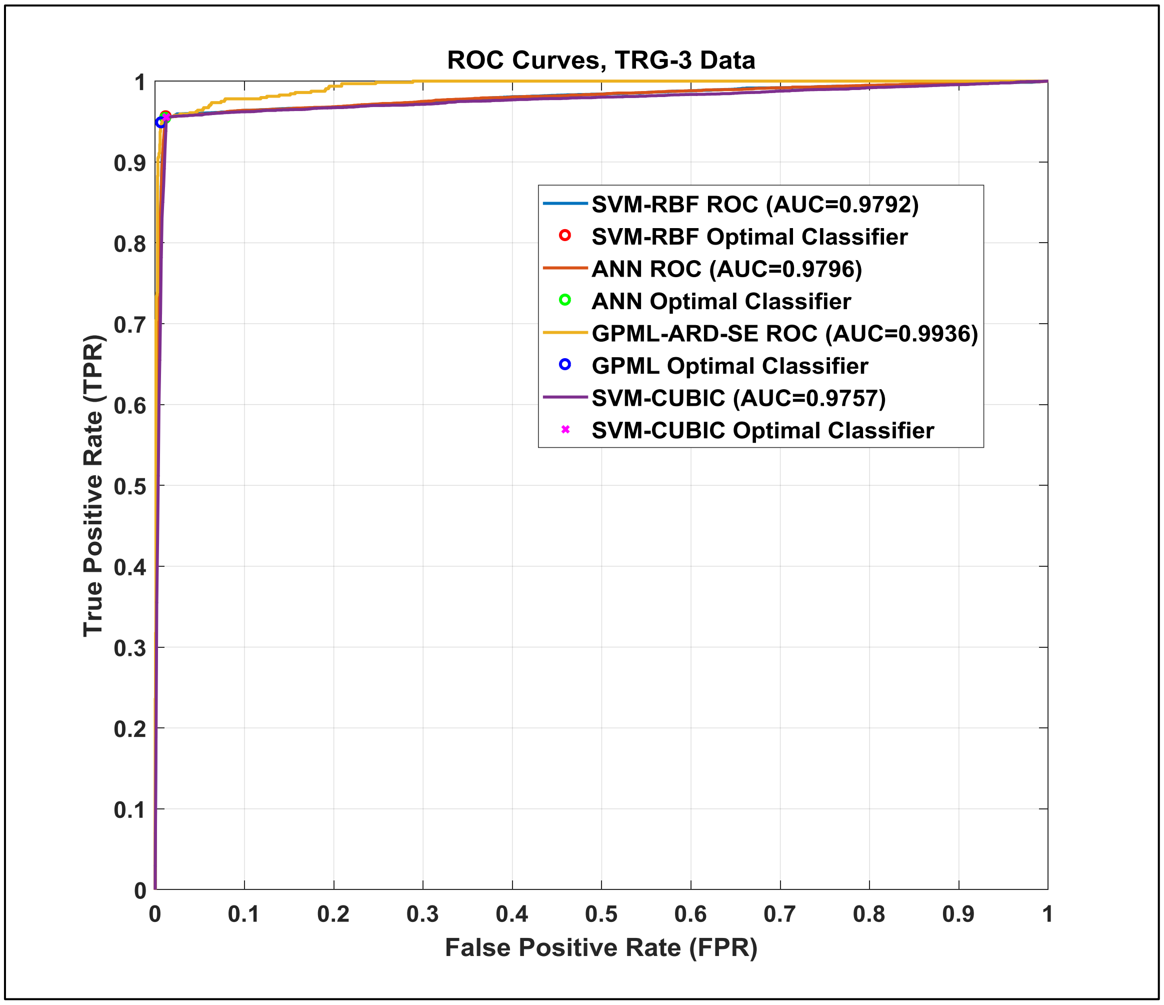


Figure 85: ROC analysis of top four performing classifiers during test (TRG-3 data).

The ROC curve in Fig. 85 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.9792, 0.9757, 0.9796, and 0.9936 respectively, all of which indicate excellent, almost perfect classifier performance. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. The SVM classifier outputs the best guess label 0 or 1 (no upset/upset).

Classifier Performance During Testing – TST-3 Data

To build the test performance matrix shown in Fig. 87 and the training performance matrix shown in Fig. 84, the probabilistic output of the ANN and the GP needs to be converted to a binary (upset/no upset) output. To do this, a decision boundary (threshold) needs to be determined above which the probability of upset is set to 1 and below which the probability of upset is set to 0 without compromising the accuracy of the machine. Fig. 86 shows a plot of prediction accuracy vs. threshold for TST-3 data and it indicates that reasonable threshold choice is 0.5, without compromising prediction accuracy.

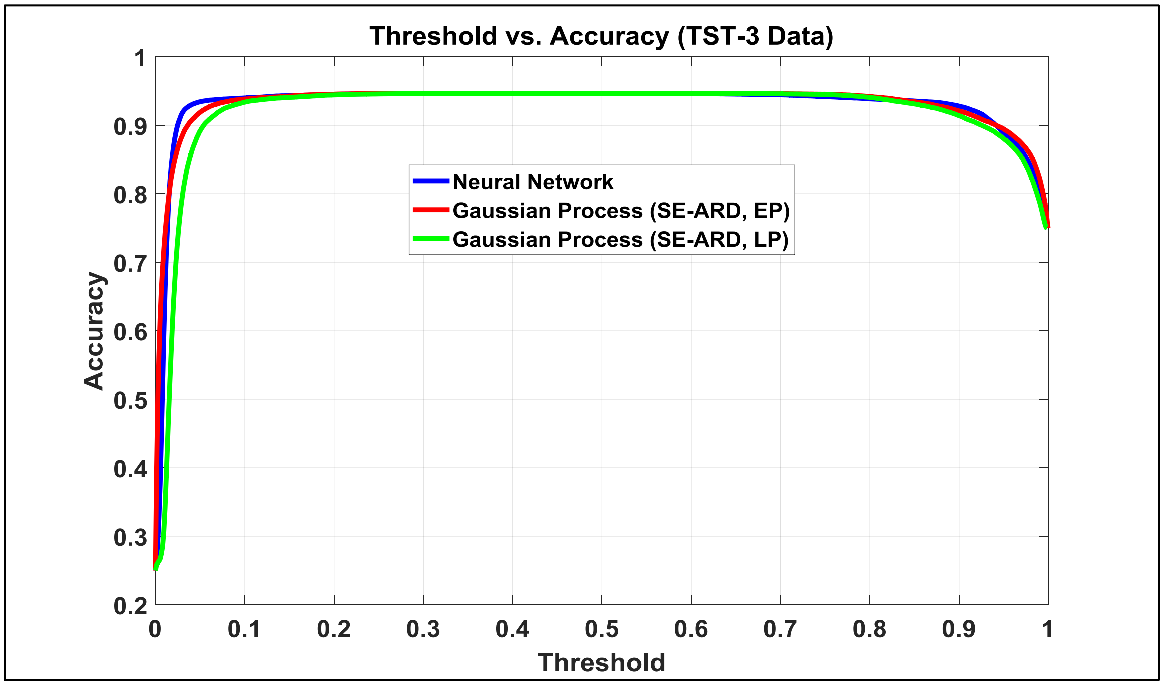


Figure 86: Threshold vs. accuracy for the TST-3 dataset.

The TST-3 test case contains 38,300 test points that will be used to assess the classifier generalization performance. As shown in Fig. 87, on the classification confusion matrix for the TST-3 test case, the SVM-RBF and the GP-SE-ARD classifier performed best, both of which have an overall accuracy of 94.7%, followed by the ANN and SVM-CUBIC classifiers with accuracies of 94.6% and 94.5%, respectively. The test performance of all classifiers was slightly worse than performance during training.

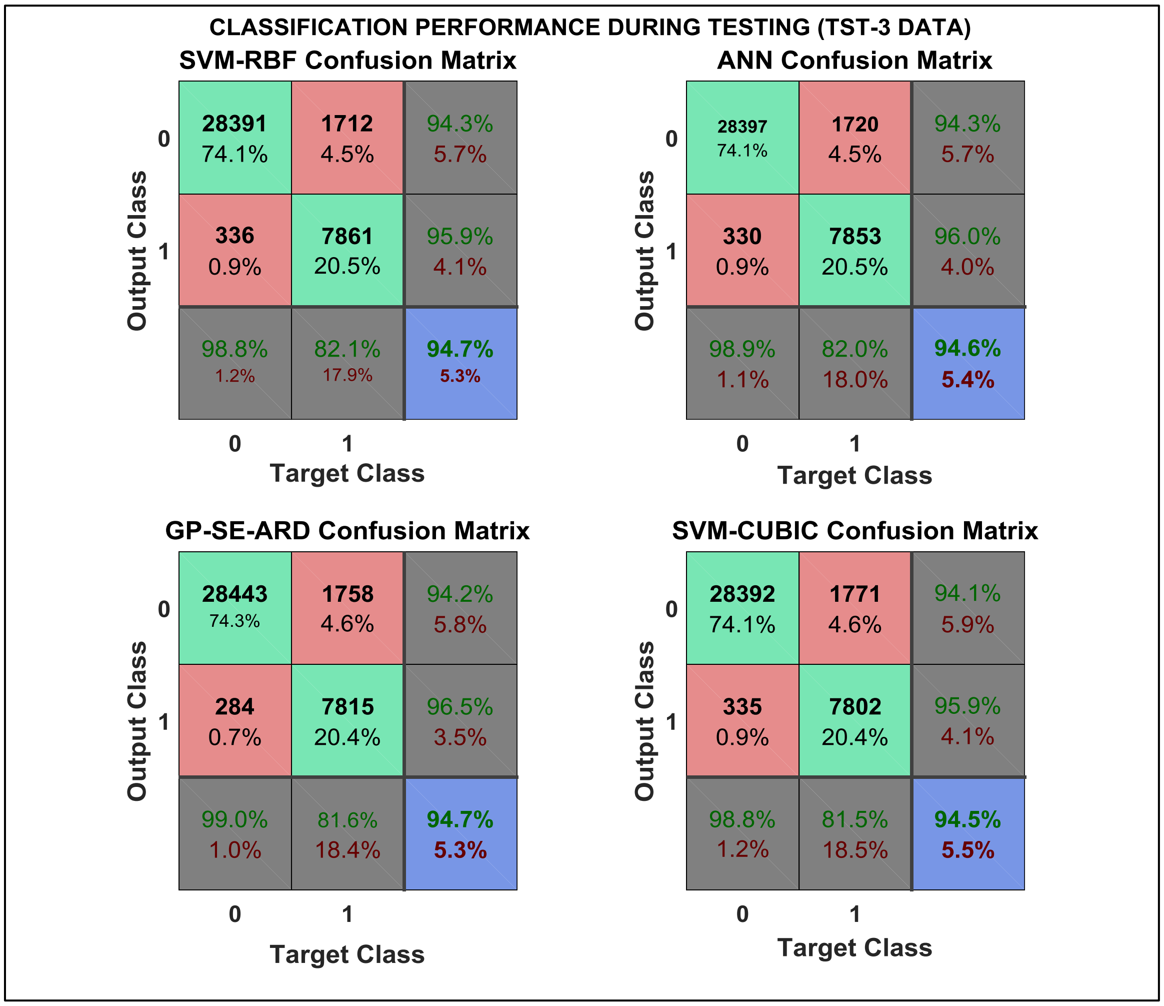


Figure 87: Classifier performance during testing (TST-3 dataset).

The ROC curve in Fig. 88 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.9099, 0.9098, 0.9085, and 0.9073, all of which are in excellent agreements with the respective training AUCs. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset.

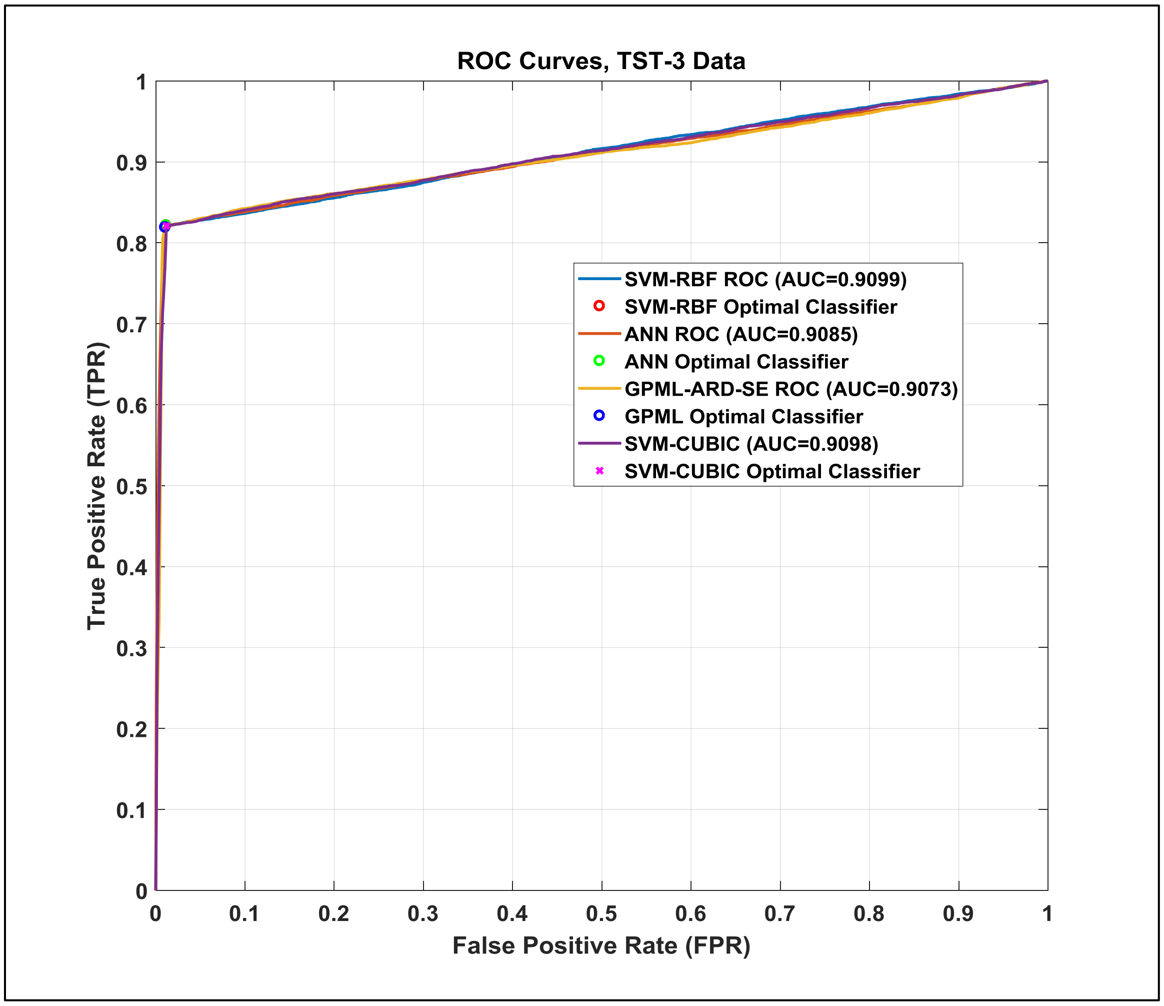


Figure 88: ROC analysis of top four performing classifiers during test (TST-3 data).

The PoE plot illustrates how the excellent accuracy during training and generalization translates to the classifiers' ability to approximate the PoE as a function of RF pulse injection location for any given injected RF power level, frequency, and pulse width. Just as in the previous two data sets, the PoE plot for the TST-3 case in Fig. 89 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points, via the following formula

$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}\!/_{\text{Number of Experiments}}$. The SVM-RBF classifier approximates the PoE curve for each clock edge extremely well. As was the case in previous test cases, most of the misclassification contained in the error rates stem from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. Fig. 89 shows how the excellent training and test performance translates to the approximation of the PoE plot as a function of injection location for a given injected RF power level, pulse width, and frequency.

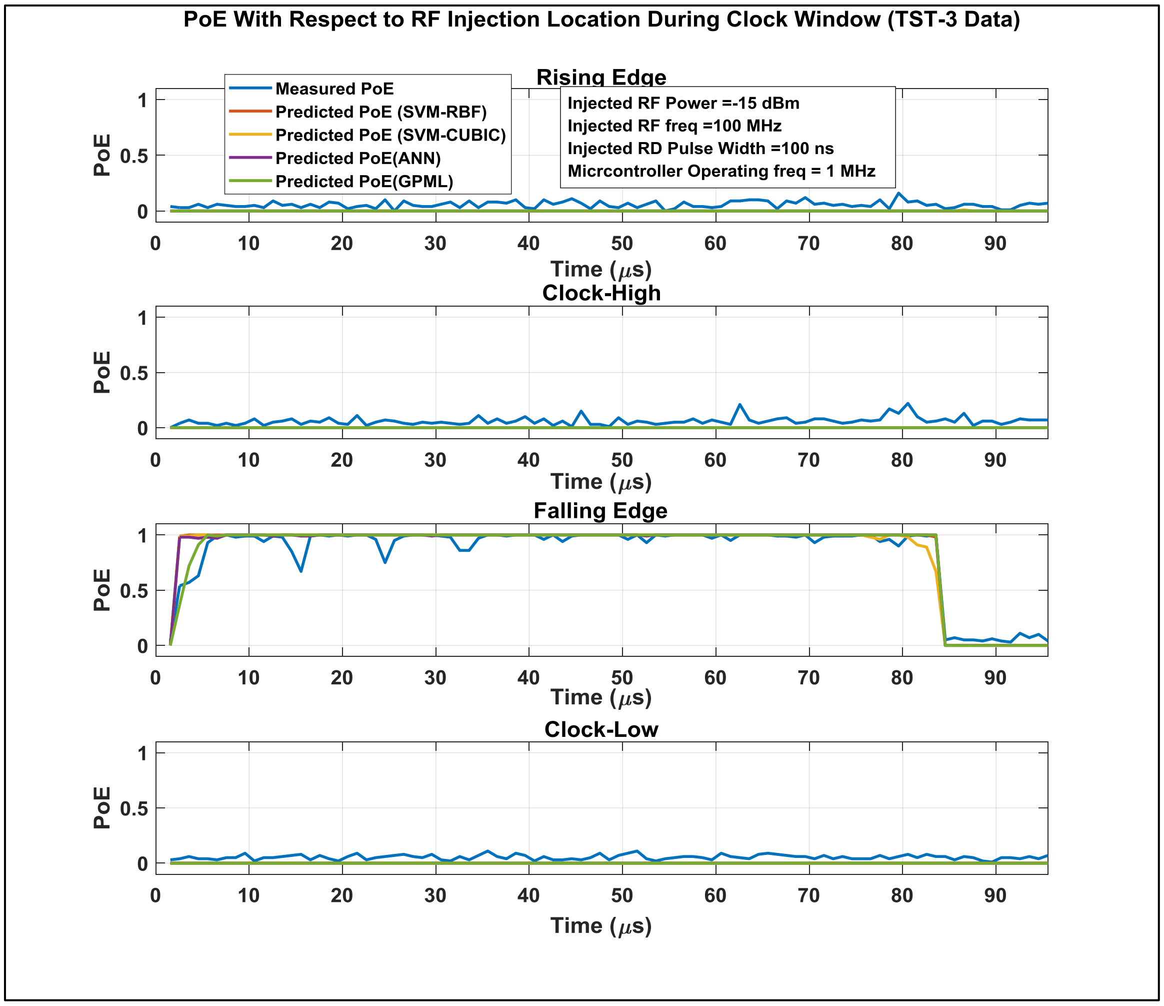


Figure 89: PoE with respect to RF injection location during clock window (TST-3 data).

To summarize the TE-3 test case, all three machine learning methods (SVM, ANN, and GPML) performed very well on TE-3 data, which consisted of 38,300 training points (TRG-3) and 38,300 test points (TST-3). The average training error across all three methods is 0.0218 (2.18%), meaning that they were almost 97.82% accurate during training, on average, with the actual training accuracy ranging from 94.98% for the SVM-LIN classifier to 98.3% for the GP-SE-ARD classifier. Performance on the test data (TST-3) was also excellent. The average error across all classifiers for the test data is 0.0592 (5.92%), which is higher than the average training error of 2.18%. The best overall performing machine for TE-3 data is the support vector machine with a radial basis function kernel (SVM-RBF), even though the Bayes optimization of the model parameters and the training to determine the support vectors took the longest to determine when compared to training times among the three methods. The execution time for training and test is not an objective of the research, and thus only a cursory qualitative assessment is provided. Classifier performance for the TE-3 dataset is summarized in Table 14. The overall measured PoE for the TE-3 test case is 0.2499, with the machine-approximated values being 0.2140, 0.2138, and 0.2125 for the SVM-RBF, ANN, and GP-SE-ARD respectively.

Table 14: Classifier performance summary for the TE-3 data set.

TE-3 Data (TRG-3(N)=38,300 / TST-3(N)=38,300, TOTAL(N)=76,600) Injected RF Frequency $(f) = 100$ MHz Injected RF Power $(P) = -20$ dBm Injected RF Pulse Width $(PW) = 100$ ns		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel $C = 4.0621,\ \sigma = 2.0760$ Bayes Optimization Est. Error: 0.0501	0.0502	0.0822
SVM – 2nd Degree Polynomial (Quadratic) Kernel $C = 5.3501e+03, \quad \sigma = 4.2421$ Bayes Optimization Est. Error: 0.0223	0.0224	0.0570
SVM – 3rd Degree Polynomial (Cubic) Kernel $C = 1.2546e-04, \quad \sigma = 486.4421$ Bayes Optimization Est. Error: 0.0196	0.0200	0.0550
SVM – Radial Basis Function (RBF/Gaussian) Kernel $C = 7.4541, \quad \sigma = 0.9517$ Bayes Optimization Est. Error: 0.0192	0.0192	0.053
Artificial Neural Network (ANN)	0.019	0.053
Gaussian Processes for Machine Learning (GPML) Squared-Exponential Covariance Function w/ARD N=3,000, T=38,300	0.0187	0.055
Machine Learning Technique	**Overall PoE (TST-3)**	
	Measured	**Predicted**
SVM - RBF	0.2499	0.2140
ANN		0.2138
GPML – SE-ARD		0.2125

Chapter 5-4 Test Experiment 4 (TE-4) Data Analysis

The data for test experiment 4 (TE-4) is also divided into two sets. The first set of 38,300 data points is used to train SVM and ANN and select a model that provides the best fit, given the data (TRG-4 data). The GP will be trained on a randomly selected TRG-4 subset of 3,000 data points. The reason for the reduction in the training data set for the GP model from 5,000 to 3,000 was done to reduce training time, without compromising prediction accuracy. The quantitative analysis of training/prediction times is not within the scope of this research, as the primary assessment criterium for classifier performance is prediction accuracy. The second subset of the TE-4 data set consists of 38,300 test data points (TST-4 data). The TST-4 dataset will be used to test the machine and generate the TST-4 confusion matrix, TST-4 ROC curves, and the TST-4 PoE plot. None of the machines under consideration have been exposed to the TST-4 data set.

For the TE-4 data set, the injected RF frequency is $f = 50$ MHz, the injected RF power level is $P = -15$ dBm, and the injected RF pulse width is $PW = 50$ ns. The RF pulse is injected into the clock pin, and it is timed to coincide with each clock edge (rising edge, clock-high, falling edge, clock-low) starting at time $t = 0.5\mu s$ until $t = 95\mu s$, which is the time window during which the MCU initializes and completes the counting operation. In sum, there are 383 injection locations during the clock window, each consisting of 100 shots (instance during which RF is injected into the clock pin), for a total of 38,300 data points. The first test run is used to collect 38,300 data points that will be used to train each machine, except the GP, which will be trained with a randomly selected subset of 3,000 data points. The test is repeated to collect an additional 38,300 data that will be used to test each trained machine.

The machines under consideration include SVMs with linear (SVM-LINEAR), quadratic (SVM-QUADRATIC), cubic (SVM-CUBIC), and Gaussian/radial basis function (SVM-RBF) kernels. The other two models used to model MCU upset include a two-layer feedforward neural network (ANN) that is trained using the scaled conjugate gradient and GPs with the squared exponential covariance functions with automatic relevance determination (GP-SE-ARD).

The performance of each machine is assessed during training and during the test phase. The training assessment features a Bayesian optimization analysis for the selection of key SVM parameters, a confusion matrix for training data, and ROC analysis for training data for all classifiers. The test assessment features a confusion matrix for test data, ROC analysis for test data, and a plot of the PoE as a function of injection location, injected RF power, injected RF frequency, and injected RF pulse duration for all classifiers.

Classifier Performance During Training – TRG-4 Data

In classification and regression scenarios, SVMs require that the kernel function (with its scale) be chosen along with the misclassification penalty parameter C. One of the most effective ways to do this is to utilize Bayesian optimization, as described in section 3-5. The objective of Bayesian optimization here is to choose C and σ that will minimize the cross-validation error. Even though four different kernel types were analyzed for the SVM models, only the two best-performing SVM models will be presented in more detail for each test case, but the performance of all four kernels selections is summarized in Table 15 at the end of this section.

For the training case (TRG-4) as well as the corresponding test case (TST-4), both of which contain 38,300 test points, the best performing SVM model is the SVM-RBF model, while the second best SVM model is the SVM-CUBIC model. The selection of kernel parameters for all SVM kernels was made using a Bayesian optimization method for which the objective function was the minimum 10-fold cross-validated loss, which for the SVM-RBF resulted in $C = 33.5612$ and $\sigma = 0.6697$ with a Bayes-estimated CV-error of 0.2330 (23.30%), as shown in Fig. 90**.**

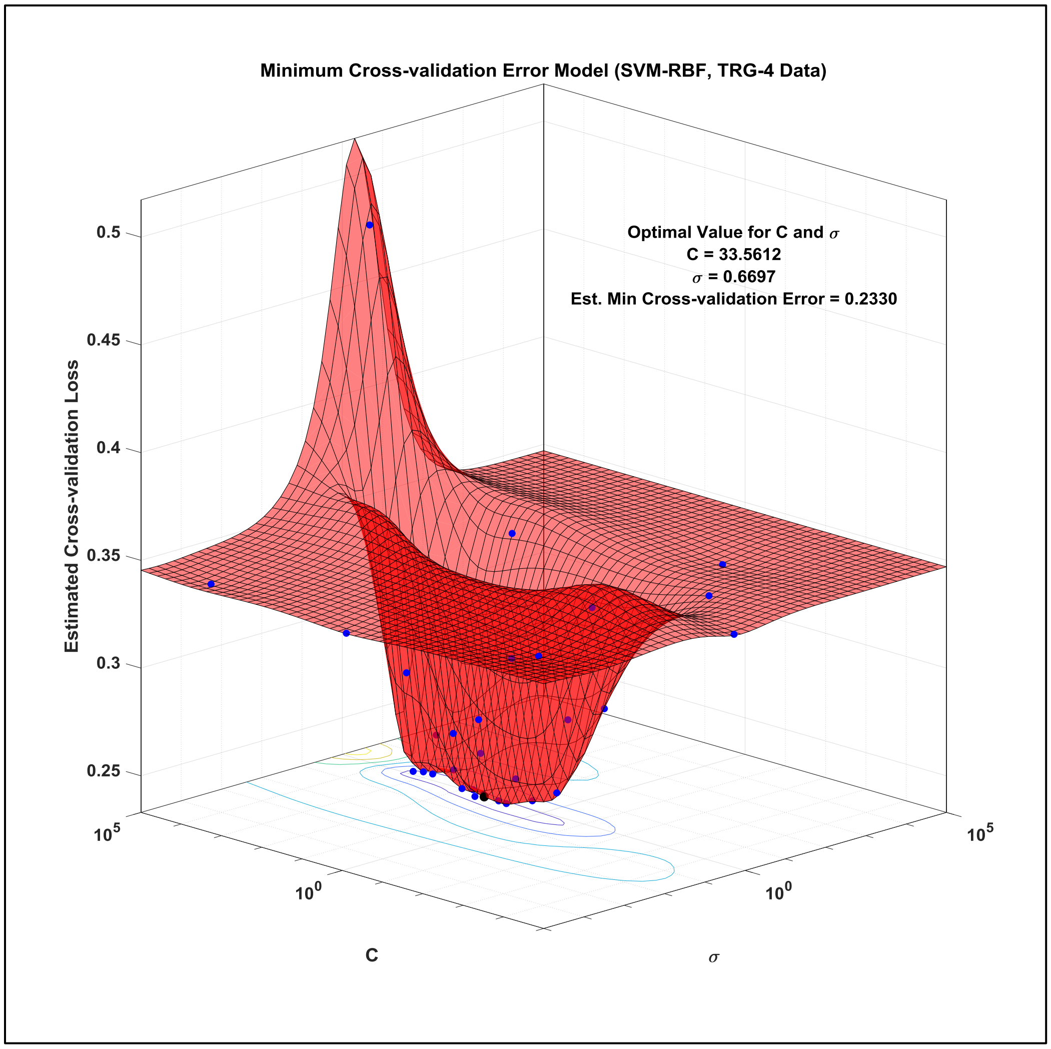


Figure 90: Bayes-optimized minimum CV-error; selection of C and σ (SVM-RBF, TRG-4 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 91**.**

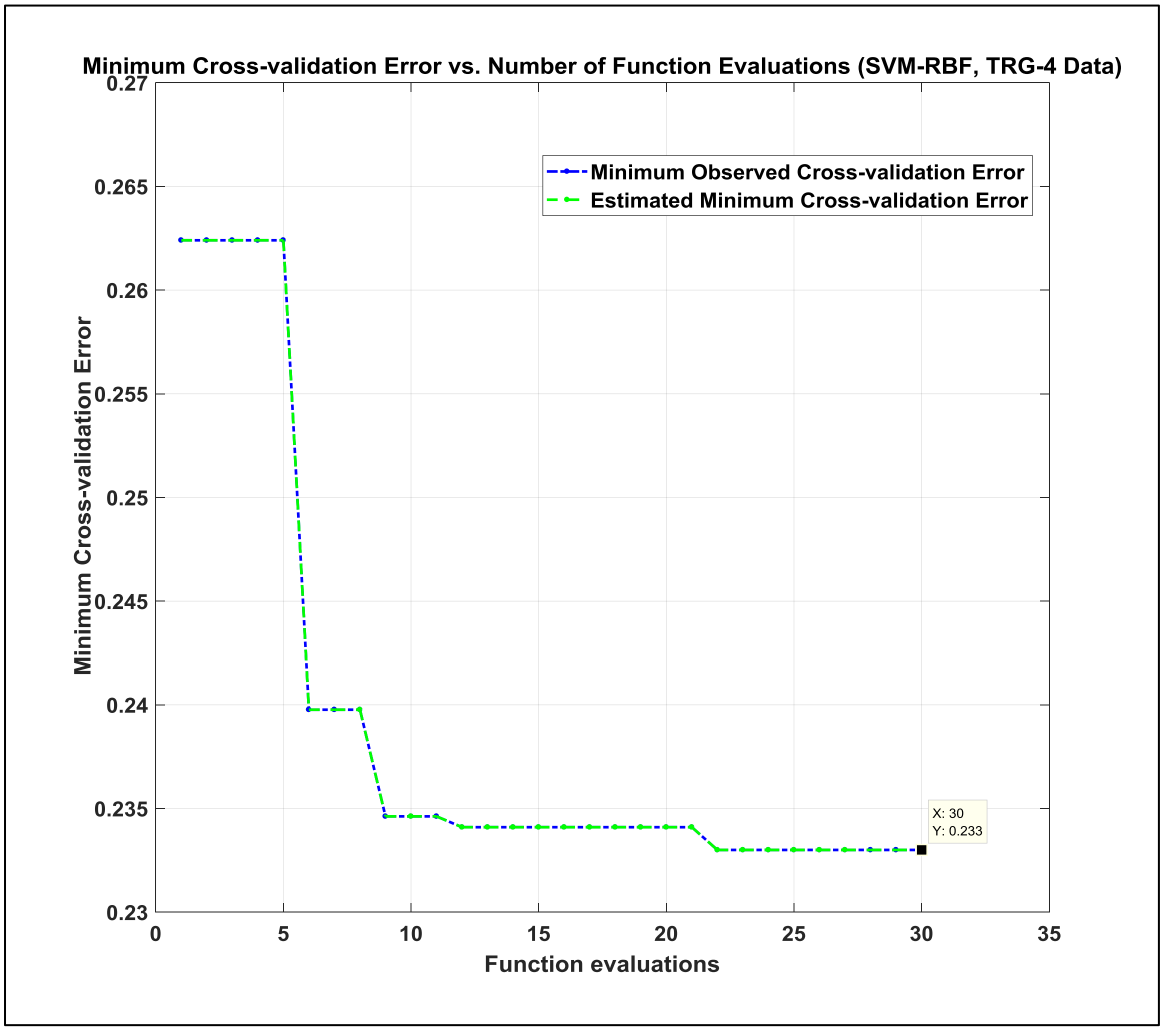


Figure 91: Cross-validation error vs. number of function evaluations (SVM-RBF, TRG-4 data).

The estimated, as well as the observed cross-validation error of 0.2330 (23.30%) for the SVM-RBF model, obtained using Bayesian optimization were in excellent agreement with the actual SVM-RBF training cross-validation error of 0.2321 (23.21%).

The Bayes-estimated error for the SVM-CUBIC kernel is 0.2585 (25.85%). The Bayes optimization of values of C and σ that minimized the cross-validation error for the SVM-CUBIC model were determined to be $C = 40.1093$ and $\sigma = 1.7520$, which is shown in Fig. 92.

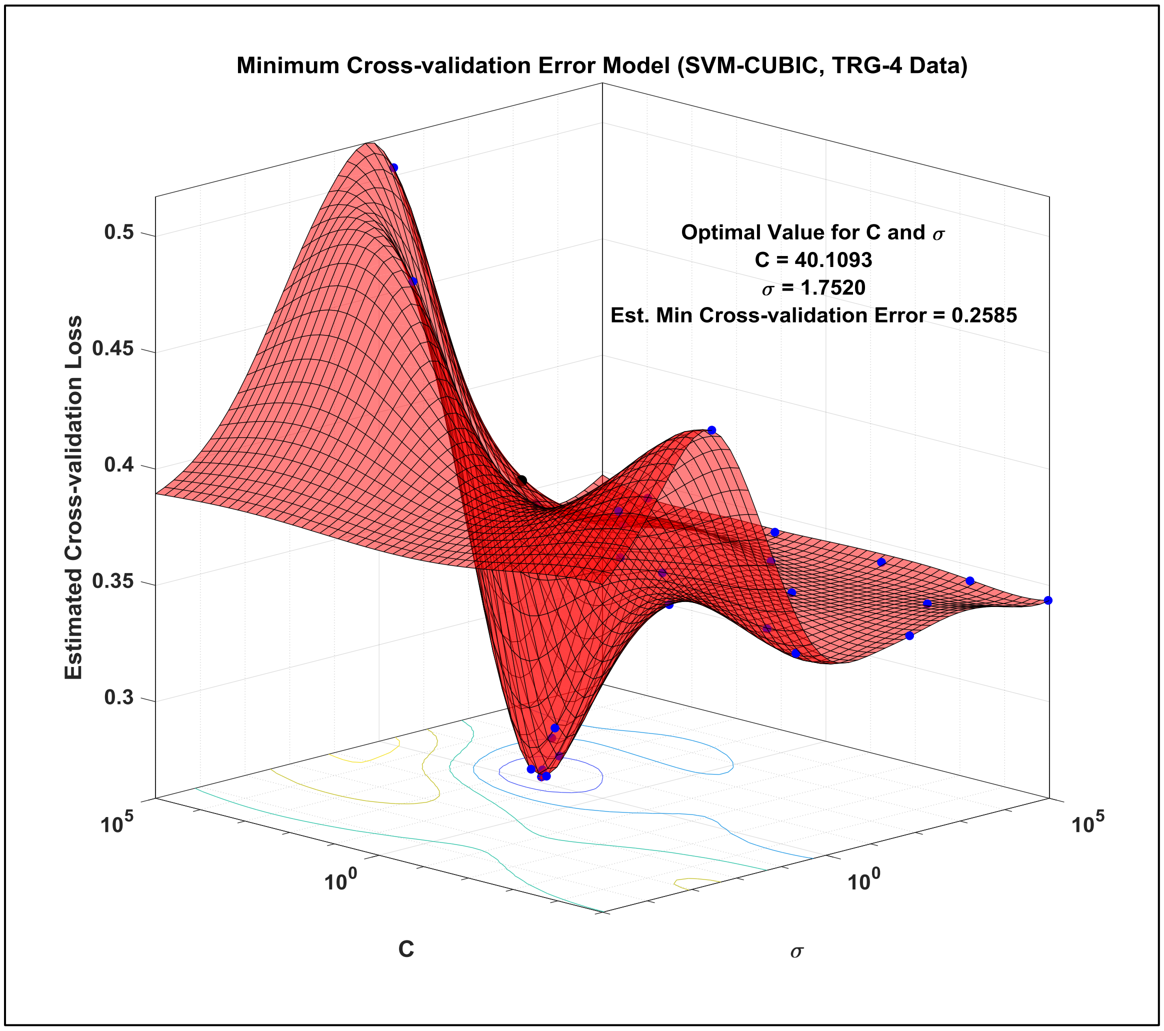


Figure 92: Bayes-optimized minimum CV-error; selection of C and σ (SVM-CUBIC, TRG-4 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 93**.**

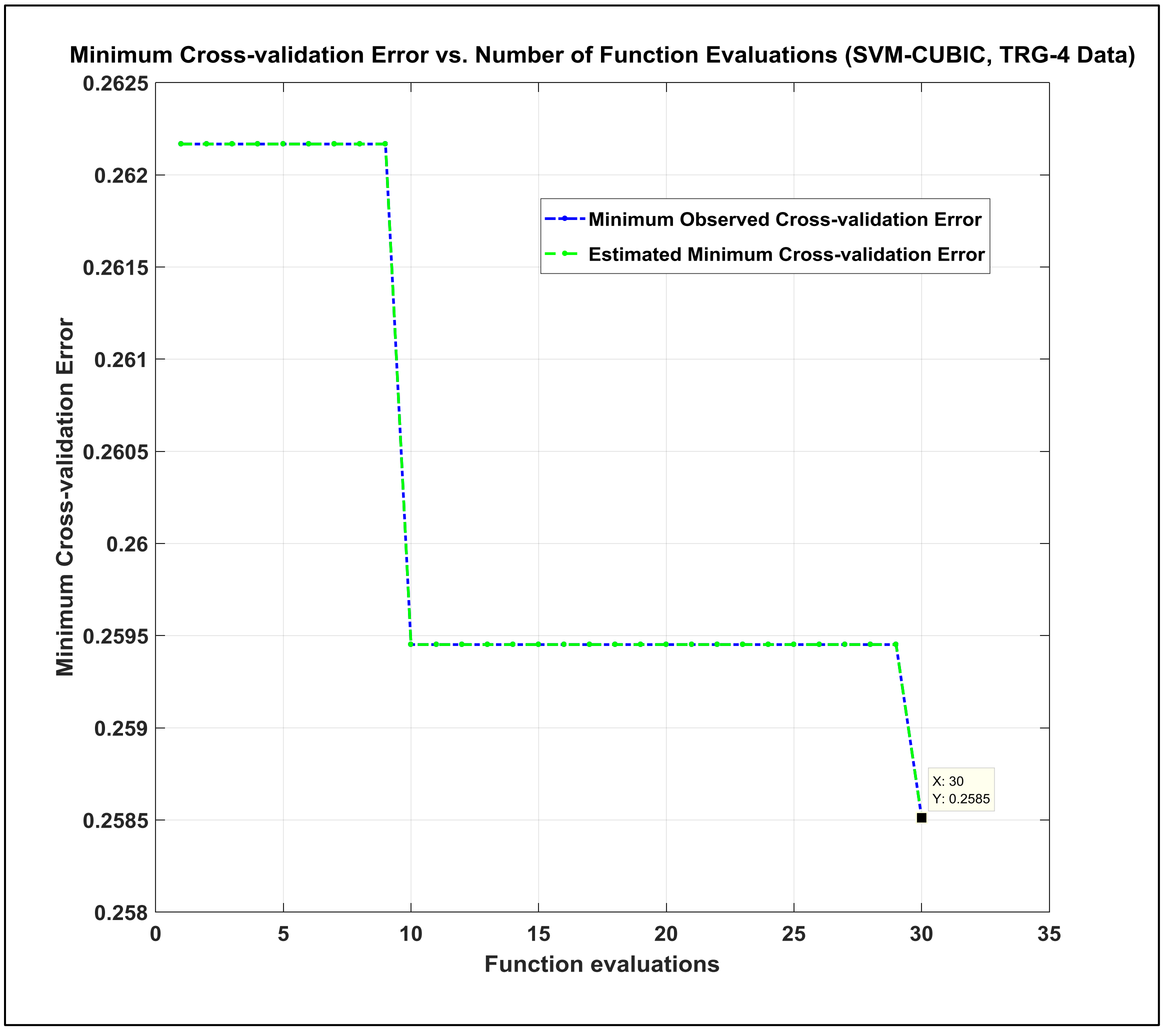


Figure 93: Cross-validation error vs. number of function evaluations (SVM-CUBIC, TRG-4 data).

The estimated, as well as the observed cross-validation error of 0.2585 (25.85%) for the SVM-CUBIC model that were obtained using Bayesian optimization, were in excellent agreement with the actual SVM-CUBIC training cross-validation error of 0.2586 (25.86%).

The confusion matrix for the training scenario (TRG-5), that contains training performance for the SVM-RBF, SVM-CUBIC, ANN, and GPML-SE-ARD is shown in Fig. 94.

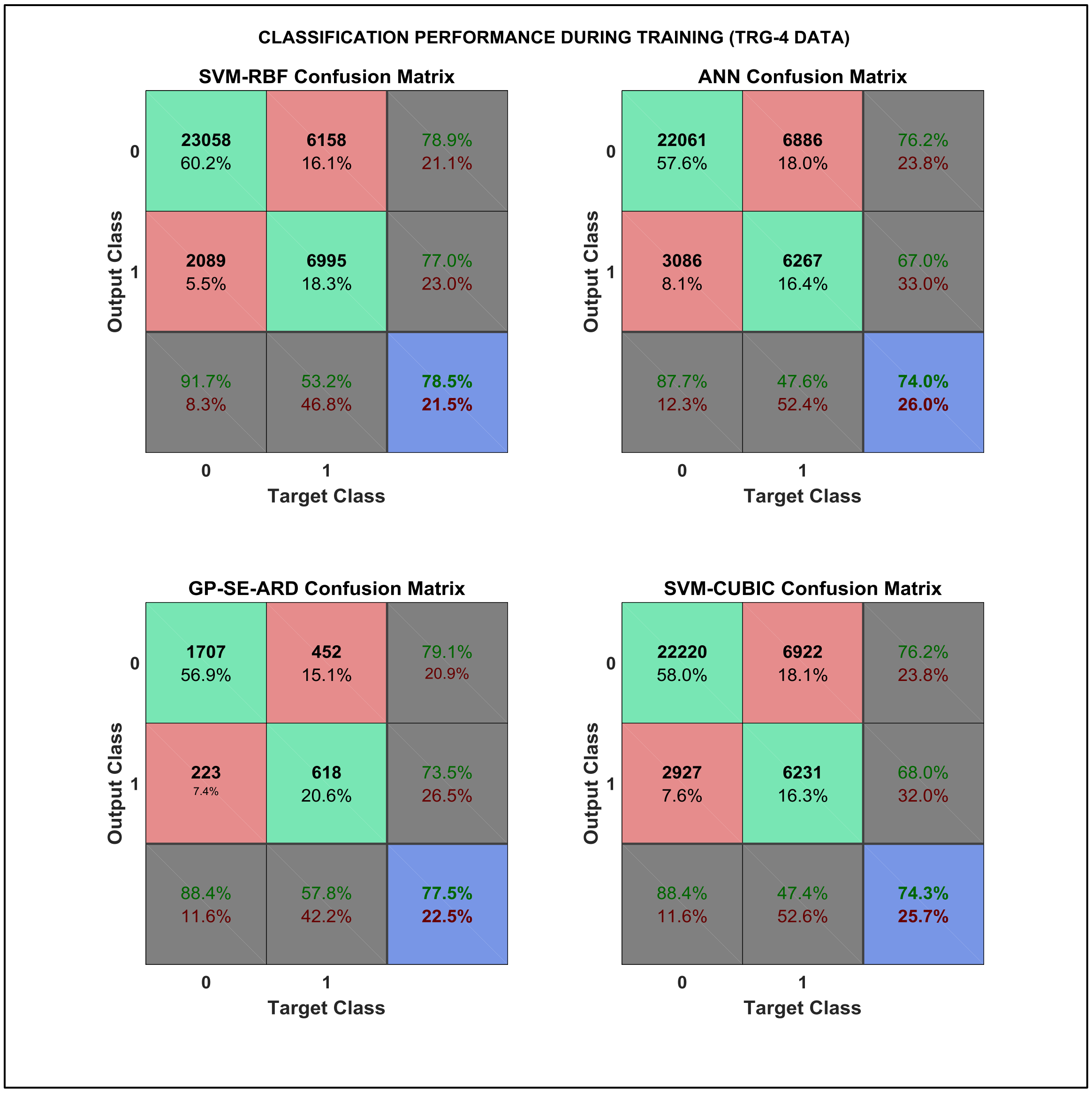


Figure 94: Classification training performance for TRG-4 data.

As shown by the confusion matrix, the SVM-RBF classifier performed best with an overall training accuracy of 78.5%, with the GP-SE-ARD and SVM-CUBIC models performing equally well with an overall training accuracy of 77.5% and 74.3%, respectively. The fourth-best classifier is the ANN model with an overall training accuracy of 74%.

The second statistical classifier description, based on the confusion matrix, is the ROC analysis. The ROC curve for all four classifiers is shown in Fig. 95**.**

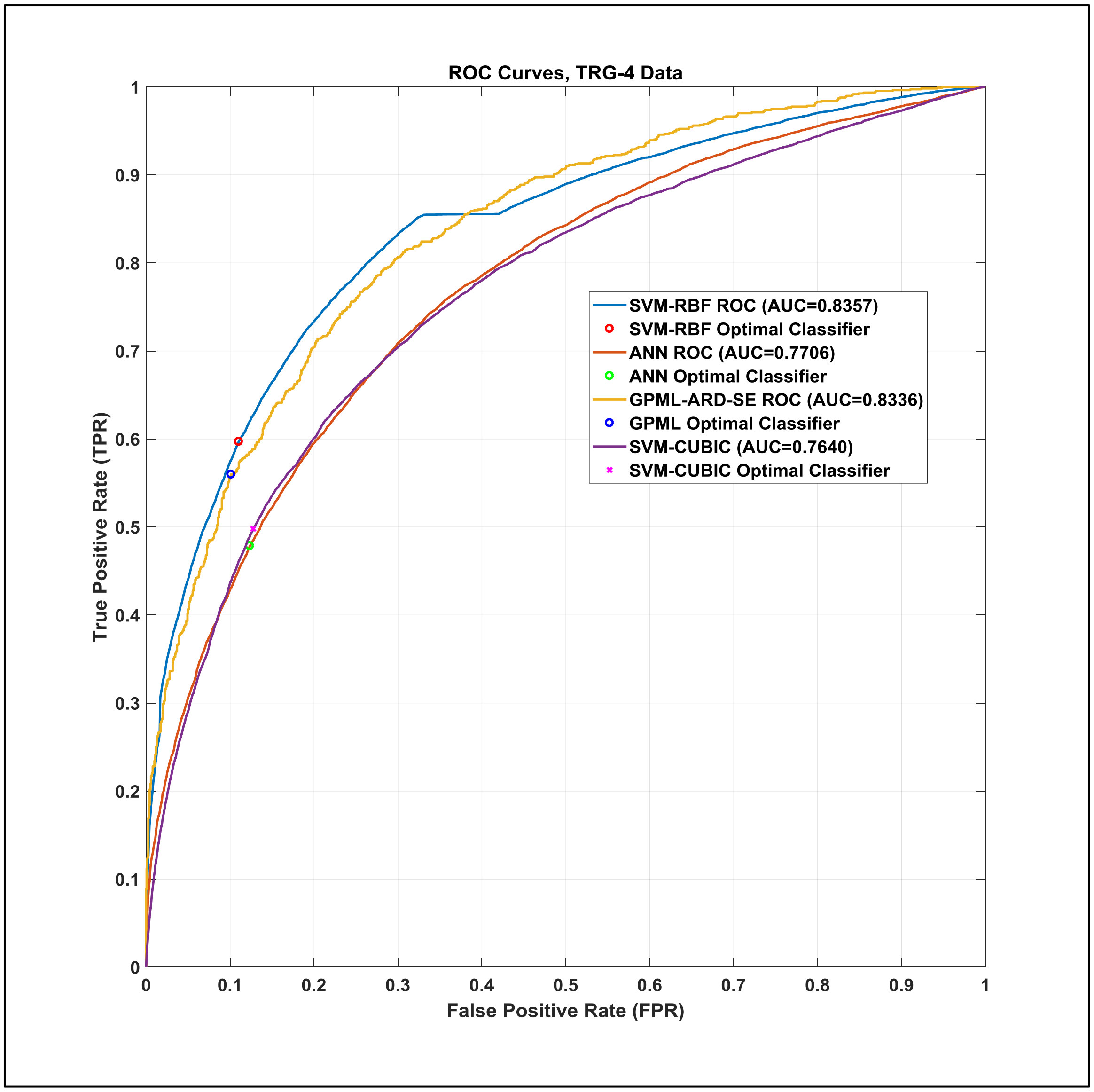


Figure 95: ROC analysis of top four performing classifiers during test (TRG-5 data).

The ROC curve in Fig. 95 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.8357, 0.7640, 0.7706, and 0.8336 respectively. According to the ROC scale introduced earlier in the chapter, the SVM-RBF and the GP-SE-ARD models represent good classifiers, while the SVM-CUBIC and ANN models represent fair classifiers. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. The SVM classifier outputs the best guess label 0 or 1 (no upset/upset).

Classifier Performance During Testing – TST-4 Data

To build the test performance matrix shown in Fig. 97 and the training performance matrix shown in Fig. 94, the probabilistic output of the ANN and the GP needs to be converted to a binary (upset/no upset) output. To do this, a decision boundary (threshold) needs to be determined above which the probability of upset is set to 1 and below which the probability of upset is set to 0 without compromising the accuracy of the machine. Fig. 96 shows a plot of prediction accuracy vs. threshold for TST-4 data and it indicates that reasonable threshold choice is 0.5, without compromising prediction accuracy.

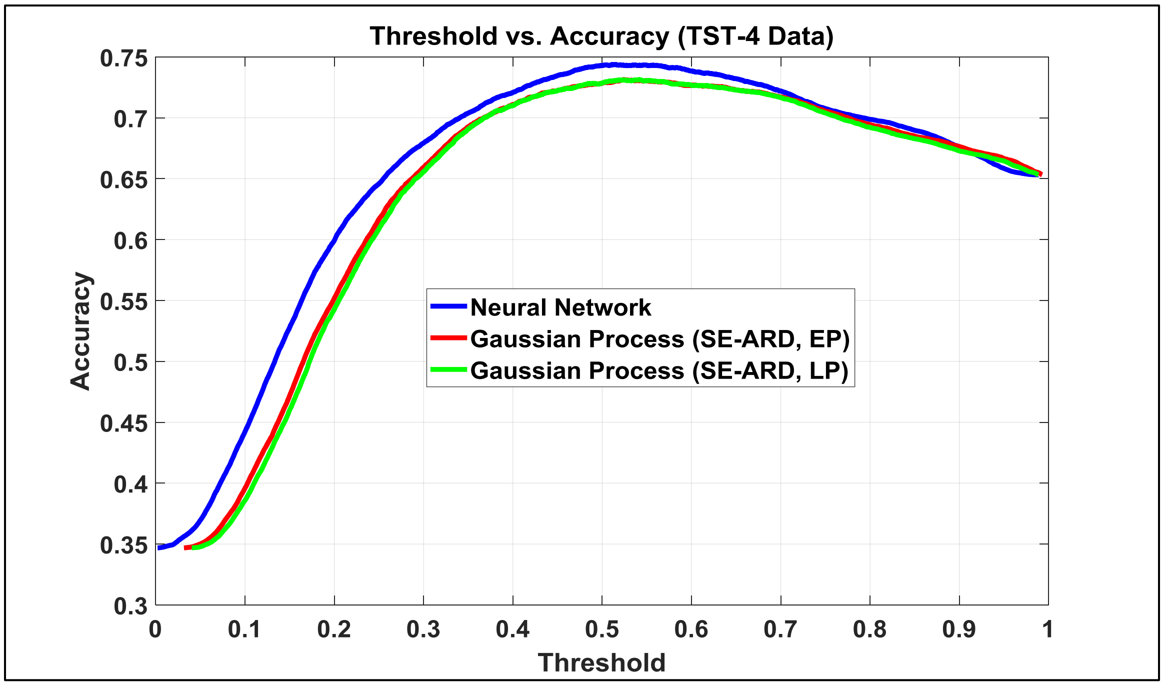


Figure 96: Threshold vs. accuracy for the TST-4 dataset.

The TST-4 test case contains 38,300 test points that will be used to assess the classifier generalization performance. As shown in Fig. 97, on the classification confusion matrix for the TST-4 test case, the SVM-RBF and the SVM-CUBIC classifier performed best, with an overall accuracy of 76.3% and 73.6%, respectively, followed by the ANN and GP-SE-ARD classifiers with accuracies of 73.4% and 73%, respectively. The test performance of all classifiers was slightly worse than performance during training.

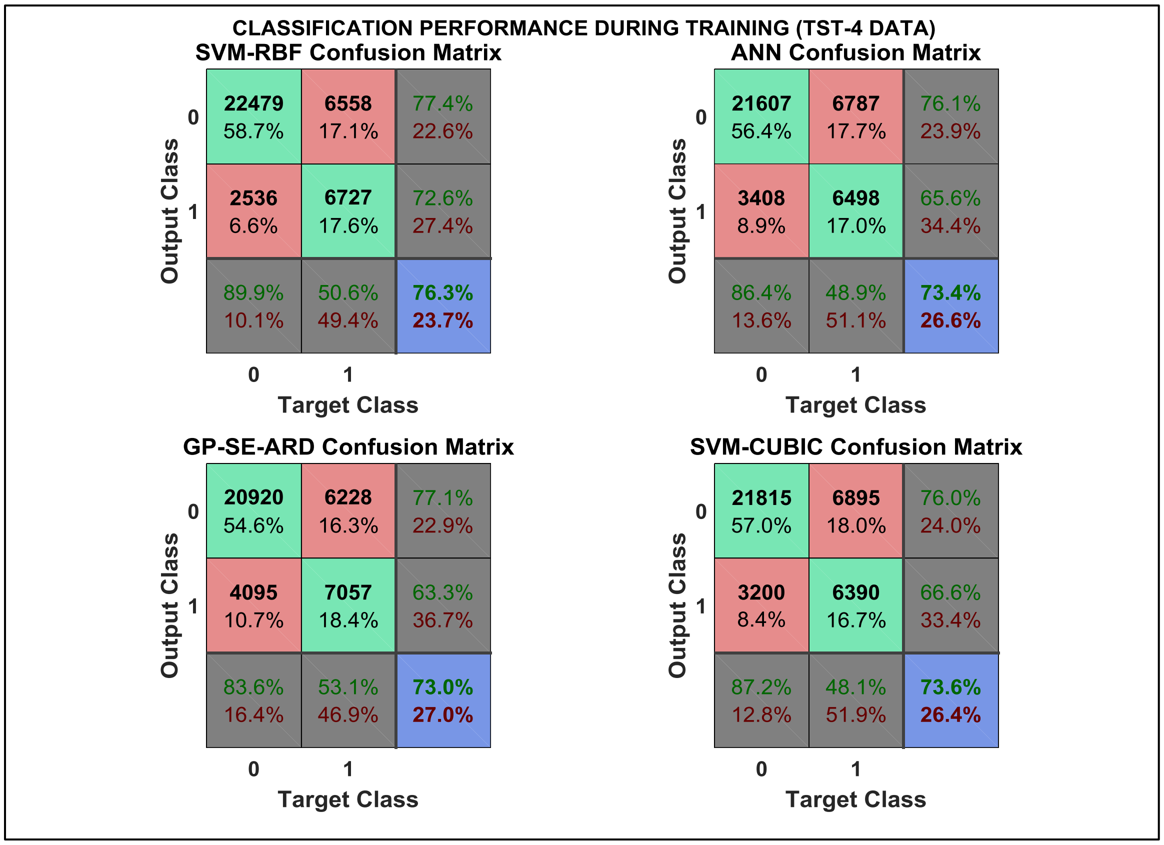


Figure 97: Classifier performance during testing (TST-4 data set).

The ROC curve in Fig. 98 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.6952, 0.6965, 0.7636, and 0.7655. There is a small miscorrelation between AUC and classifier performance in the confusion matrix partly because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset. Lower AUC scores also reflect the poorer performance during testing on unseen data. Even though their accuracies were comparable, AUC scores for the SVM classifiers indicate that SVMs represent poorly performing classifiers for this data set, while ANNs and GP-SE-ARD classifiers represent fair performing classifiers for the TST-4 data set.

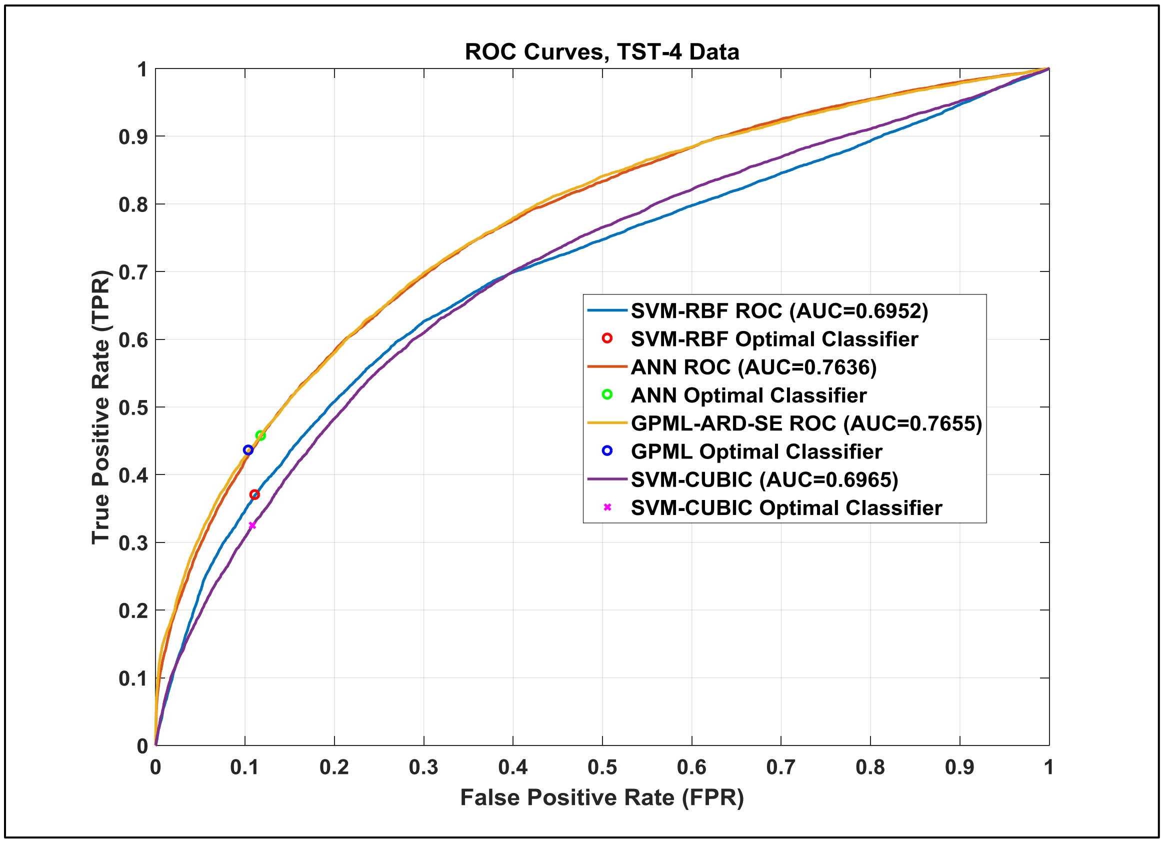


Figure 98: ROC analysis of top four performing classifiers during test (TST-4 data set).

The PoE plot illustrates how the excellent accuracy during training and generalization translates to the classifiers' ability to approximate the PoE as a function of RF pulse injection location for any given injected RF power level, frequency, and pulse width. Just as in the previous two data sets, the PoE plot for the TST-4 case in Fig. 99 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points, via the following formula

$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}\!/_{\text{Number of Experiments}}$. The SVM-RBF classifier approximates the PoE curve for each clock edge extremely well. As was the case in previous test cases, most of the misclassification contained in the error rates stem from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. Fig. 99 shows how the training and test performance translates to the approximation of the PoE plot as a function of injection location for a given injected RF power level, pulse width, and frequency.

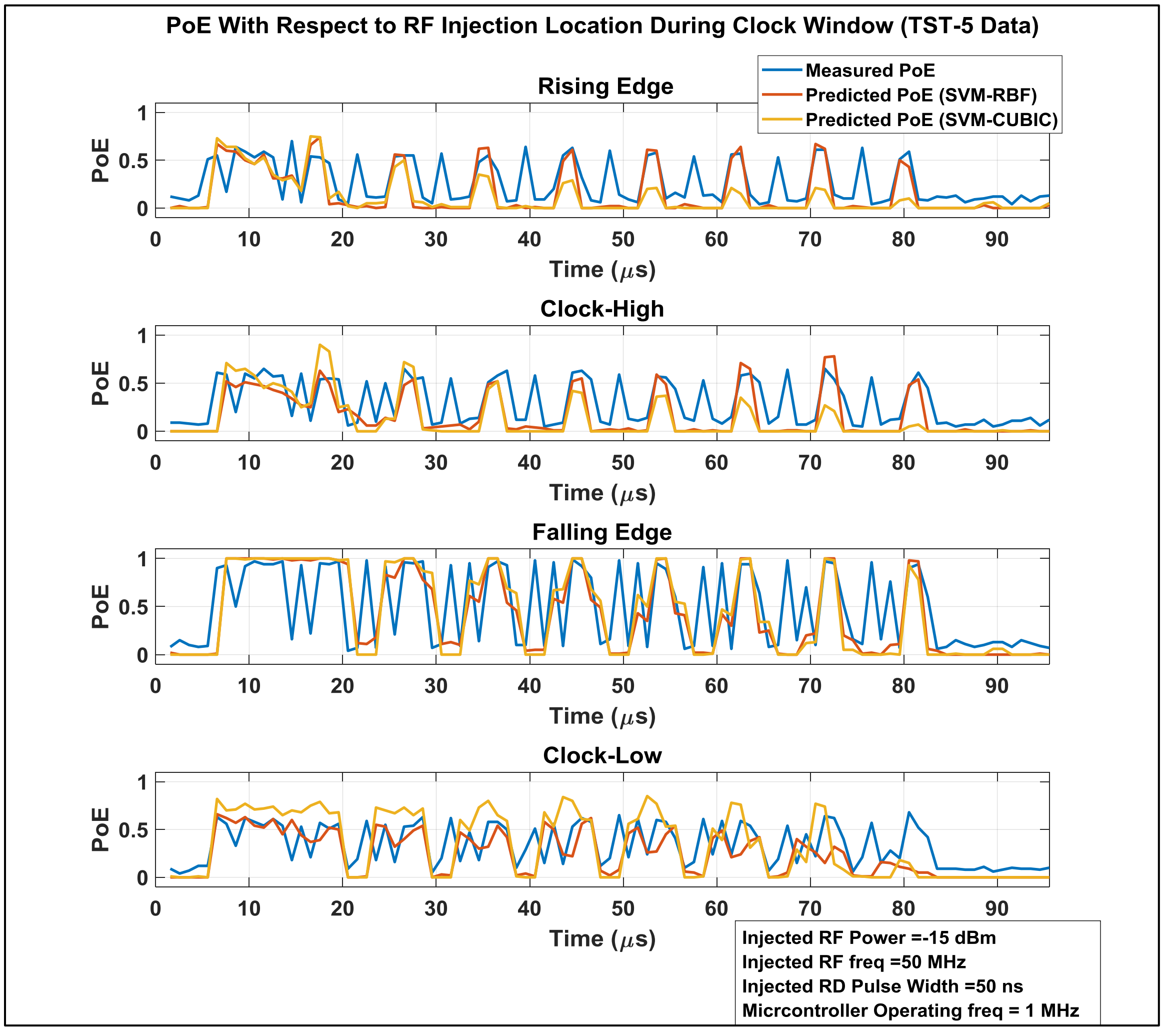


Figure 99: PoE vs. injection location for top two performing SVM classifiers (TST-4 data set).

To reduce clutter, the PoE plot for the TST-4 data set has been split to show performance for the top two SVM classifiers in Fig. 99 and the performance of the ANN and GP-SE-ARD classifiers in Fig. 100.

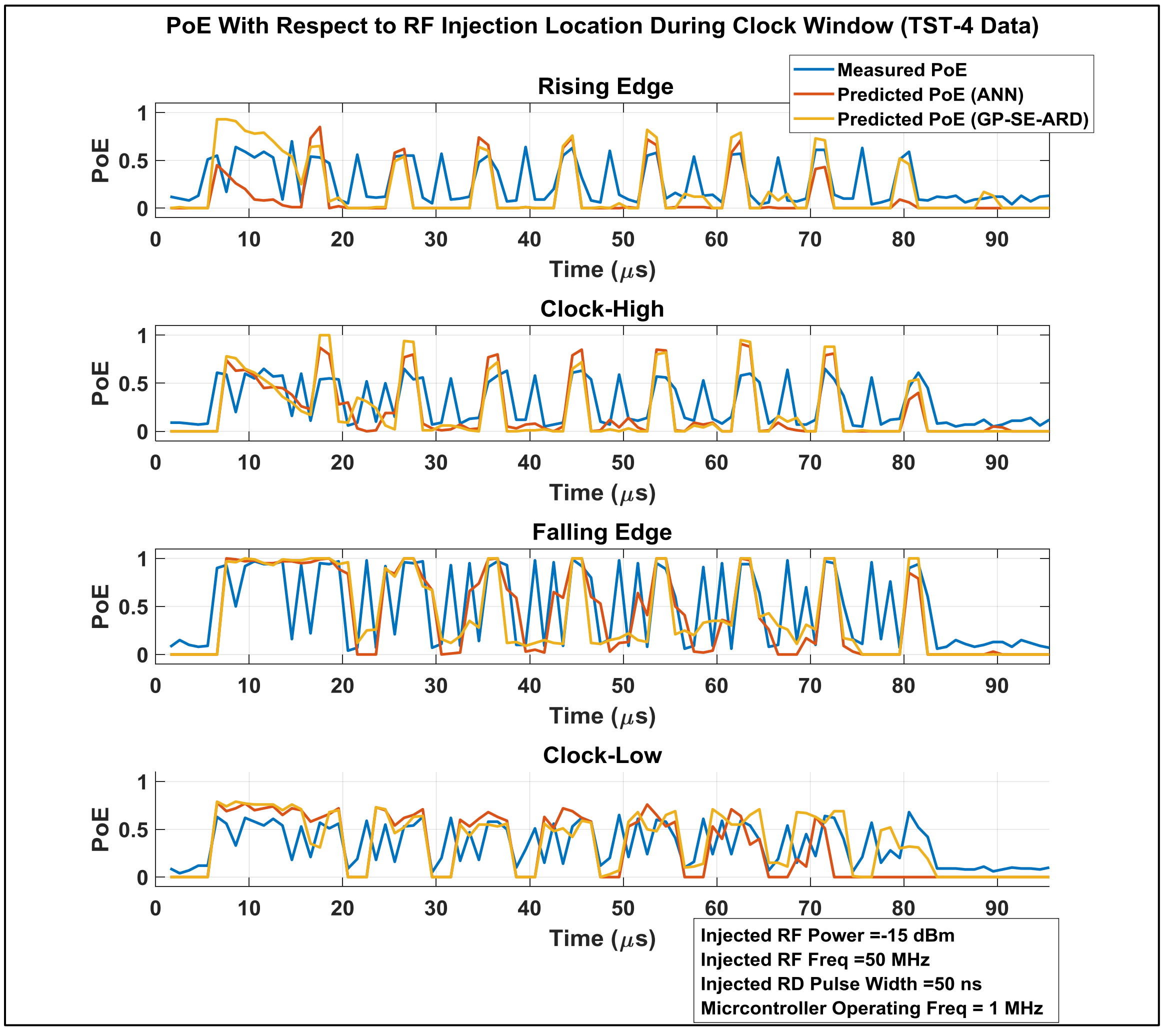


Figure 100: PoE vs. injection location for ANN and GP-SE-ARD classifiers (TST-5 data set).

To summarize the TE-4 test case, all three machine learning methods (SVM, ANN, and GPML) performed at an acceptable level on TE-4 data, which consisted of 38,300 training points (TRG-4) and 38,300 test points (TST-4). The average training error (TRG-4 data) across all three methods is 0.2654 (26.54%), meaning that they were almost 73.46% accurate during training, on average. Performance on the test data (TST-

4) was also acceptable. The average error across all classifiers for the test data is 0.2768 (27.68%), which is higher than the average training error of 26.54%. The best overall performing machine for TE-4 data is the SVM-RBF. The measurement of training and prediction times is an objective of the research. The single criterion for performance assessment is prediction accuracy. Classifier performance for the TE-4 dataset is summarized in Table 15. The overall measured PoE for the TE-4 test case is 0.3469, with the machine-approximated values being 0.2419, 0.2586, and 0.2912 for the SVM-RBF, ANN, and GP-SE-ARD respectively.

Table 15: Classifier performance summary for the TE-4 data set.

TE-4 Data (TRG-4(N)=38,300 / TST-4(N)=38,300, TOTAL(N)=76,600) Injected RF Frequency $(f) =$ 50 MHz Injected RF Power (P) = -15 dBm Injected RF Pulse Width $(PW) =$ 50 ns		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel $C = 1.5132e-04, \quad \sigma = 0.0072$ Bayes Optimization Est. Error: 0.3434	0.3434	0.3470
SVM – Quadratic Kernel $C = 2.1680, \quad \sigma = 0.5028$ Bayes Optimization Est. Error: 0.2728	0.2730	0.2770
SVM –Cubic Kernel $C = 40.1093, \quad \sigma = 1.7520$ Bayes Optimization Est. Error: 0.2585	0.2586	0.2640
SVM – Radial Basis Function (RBF) Kernel $C = 33.5612, \quad \sigma = 0.6697$ Bayes Optimization Est. Error: 0.2330	0.2321	0.2370
Artificial Neural Network (ANN)	0.2600	0.2660
GPs with Expectation Propagation Squared-Exponential Covariance Function w/ARD N=3,000, T=38,300	0.2250	0.2700
Machine Learning Method	**Overall PoE (TST-4)**	
	Measured	**Predicted**
SVM - RBF	0.3469	0.2419
ANN		0.2586
GP – SE-ARD		0.2912

Chapter 5-5 Test Experiment 5 (TE-5) Data Analysis

The data for test experiment 5 (TE-5) is also divided into two sets. The first set of 38,300 data points is used to train SVM and ANN and select a model that provides the best fit, given the data (TRG-5 data). The GP will be trained on a randomly selected TRG-5 subset of 3,000 data points. The reason for the reduction in the training data set for the GP model from 5,000 to 3,000 was done to reduce training time. The quantitative analysis of training/prediction times is not within the scope of this research, as the primary assessment criteria for classifier performance is prediction accuracy. The second set within the TE-5 data set also consists of 38,300 test data points (TST-5 data). The TST-5 dataset will be used to test the machine and generate the TST-5 confusion matrix, ROC curves, and the PoE plot. None of the machines under consideration have been exposed to the TST-5 data set.

For the TE-5 data set, the injected RF frequency is $f = 100$ MHz, the injected RF power level is $P = -20$ dBm, and the injected RF pulse width is $PW = 100$ ns. The RF pulse is injected into the clock pin, and it is timed to coincide with each clock edge (rising edge, clock-high, falling edge, clock-low) starting at time $t = 0.5\ \mu s$ until $t = 95\ \mu s$, which is the time window during which the MCU initializes and completes the counting operation. In sum, there are 383 injection locations during the clock window, each consisting of 100 shots (instance during which RF is injected into the clock pin), for a total of 38,300 data points. The first test run is used to collect 38,300 data points that will be used to train each machine, except the GP, which will be trained with a randomly selected subset of 3,000 data points. The test is repeated to collect an additional 38,300 data that will be used to test each trained machine.

The machines under consideration include SVMs with linear (SVM-LINEAR), quadratic (SVM-QUADRATIC), cubic (SVM-CUBIC), and Gaussian/radial basis function (SVM-RBF) kernels. The other two models used to model MCU upset include a two-layer feedforward neural network (ANN) that is trained using the scaled conjugate gradient and GPs with the squared exponential covariance functions with automatic relevance determination (GP-SE-ARD).

The performance of each machine is assessed during training and during the test phase. The training assessment features a Bayesian optimization analysis for the selection of key SVM parameters, a confusion matrix for training data, and ROC analysis for training data for all classifiers. The test assessment features a confusion matrix for test data, ROC analysis for test data, and a plot of the PoE as a function of injection location, injected RF power, injected RF frequency, and injected RF pulse duration for all classifiers.

Classifier Performance During Training – TRG-5 Data

In classification and regression scenarios, SVMs require that the kernel function (with its scale) be chosen along with the misclassification penalty parameter C. One of the most effective ways to do this is to utilize Bayesian optimization, as described in Section 3-5. The objective of Bayesian optimization here is to choose C and σ that will minimize the cross-validation error. Even though four different kernel types were analyzed for the SVM models, only the two best performing SVM models will be presented in more detail for each test case, but the performance of all four kernels selections is summarized in Table 16 at the end of the section.

For the training case (TRG-5) as well as the corresponding test case (TST-5), both of which contain 38,300 test points, the best performing model is the SVM-LINEAR model, while the second best is the SVM-QUADRATIC model. The selection of kernel parameters for all SVM kernels was made using Bayesian optimization for which the objective function was the minimum 10-fold cross-validated loss, which for the SVM-LINEAR resulted in $C = 3.7666$ and $\sigma = 0.0666$ with a Bayes-estimated CV-error of 0.0802 (8.02%), as shown in Fig. 101**.**

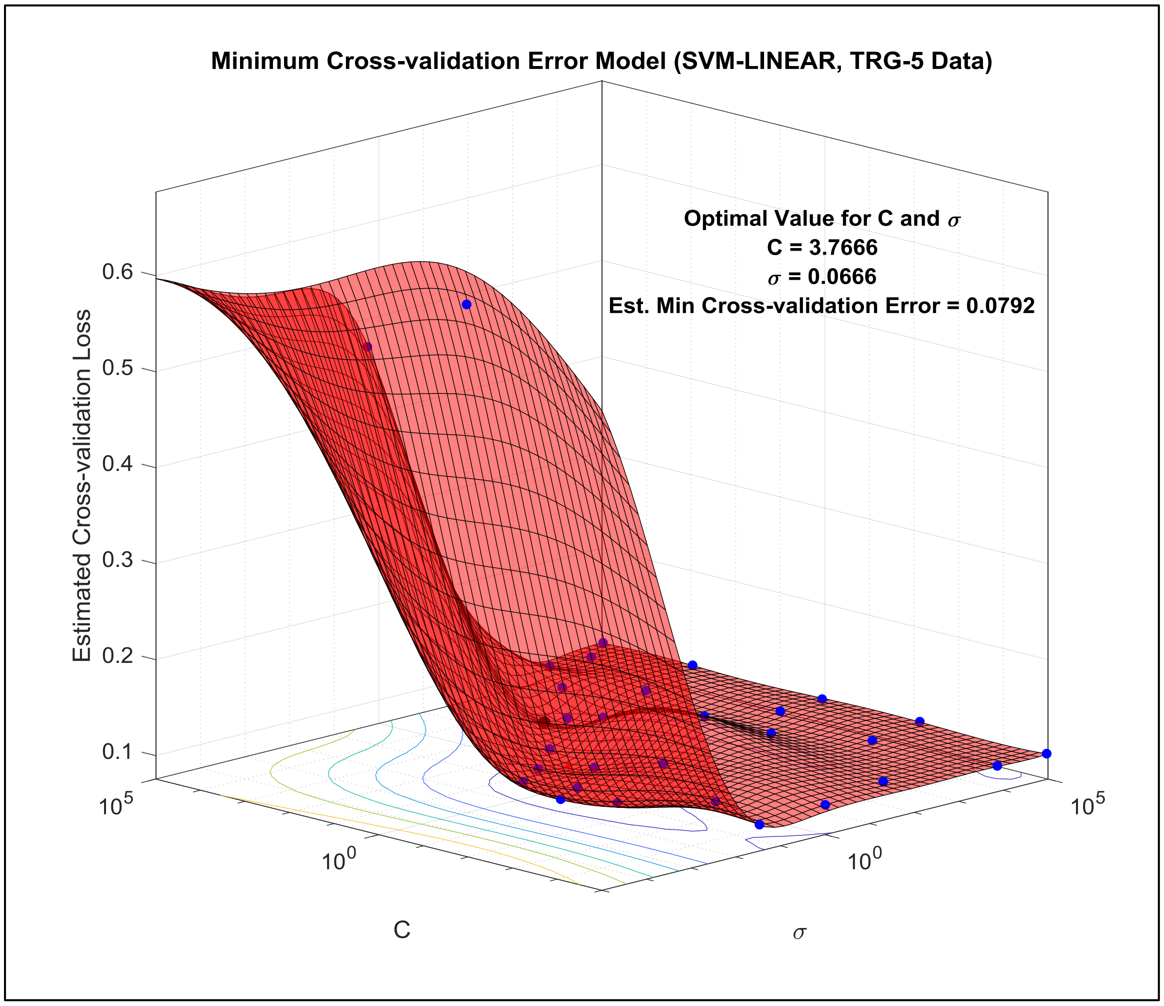


Figure 101: Bayes-optimized minimum CV-error; selection of C and σ (SVM-LINEAR, TRG-5 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 102**.**

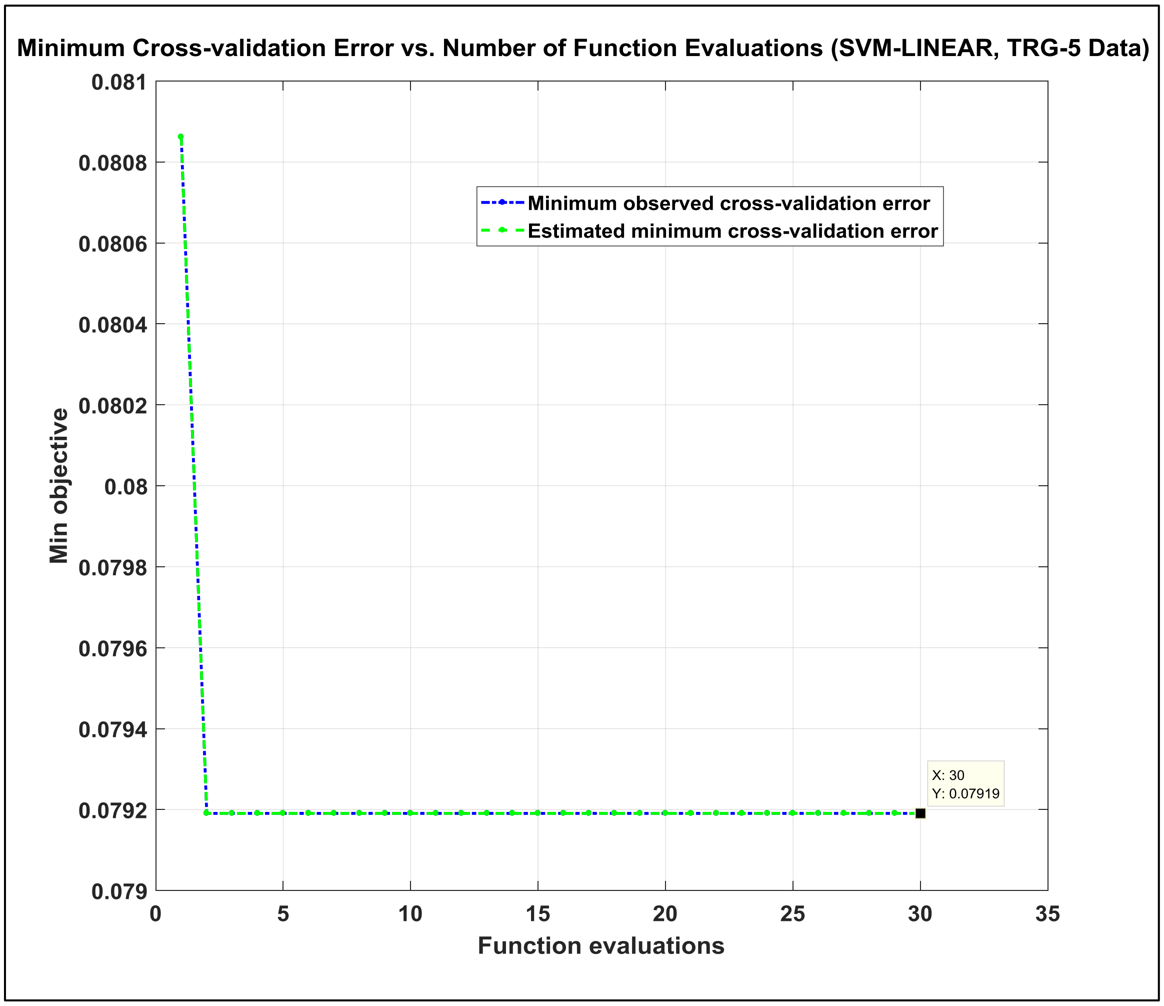


Figure 102: Cross-validation error vs. number of function evaluations (SVM-LIN, TRG-5 data).

The estimated, as well as the observed cross-validation error of 0.07919% (7.919%) for the SVM-LINEAR model, obtained using Bayesian optimization were in excellent agreement with the actual SVM-LINEAR training cross-validation error of 0.0802 (8.02%).

The Bayes-estimated error for the SVM-QUADRATIC kernel is 0.0760 (7.60%), which is excellent agreement with the actual obtained error of 0.0758 (7.58%). The Bayes optimization of values of C and σ that minimized the cross-validation error for the SVM-QUADRATIC model were determined to be $C = 9.3326e + 04$, and $\sigma = 57.0923$, which is shown in Fig. 103.

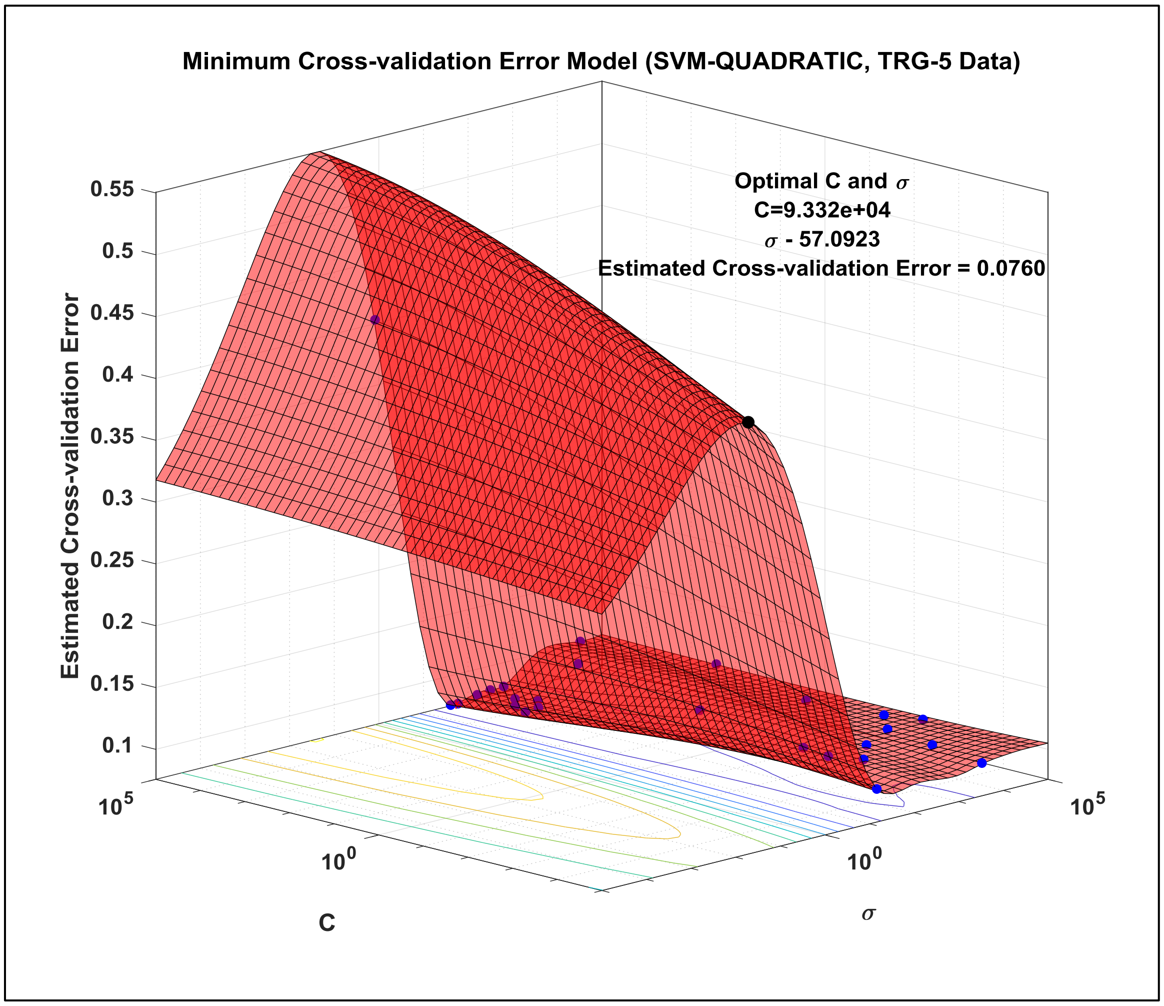


Figure 103: Bayes-optimized minimum CV-error; selection of C and σ (SVM-QUADRATIC, TRG-5 data).

In determining optimal values for C and σ that minimize the 10-fold cross-validation loss using Bayesian optimization, the number of objective function evaluations performed to reach the minimum observed and estimated error is 30, as shown in Fig. 104**.**

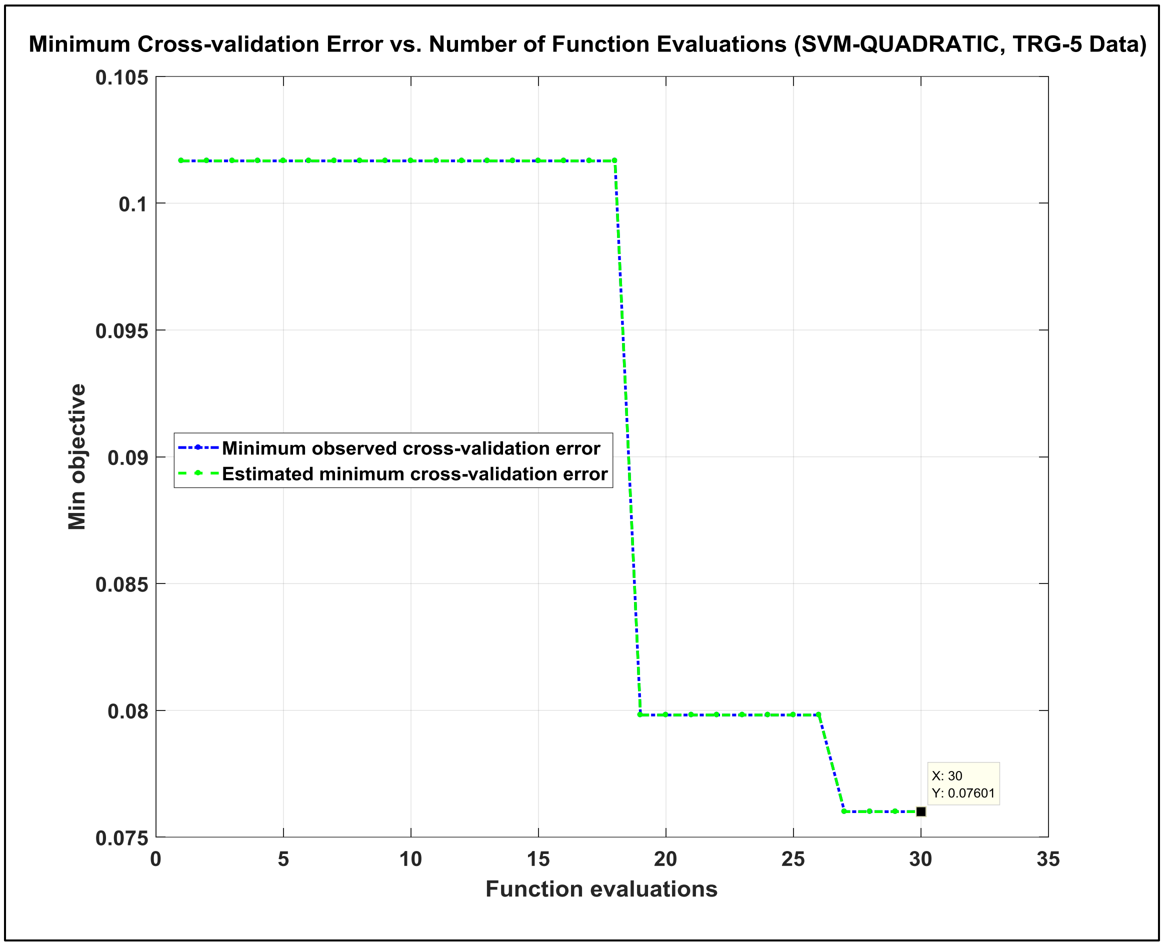


Figure 104: Cross-validation error vs. number of function evaluations (SVM-QUADRATIC, TRG-5 data).

The estimated, as well as the observed cross-validation error of 0.0760% (7.60%) for the SVM-QUADRATIC model that were obtained using Bayesian optimization, were in excellent agreement with the actual SVM-QUADRATIC training cross-validation error of 0.0758 (7.58%).

The confusion matrix for the training scenario (TRG-5), that contains training performance for the SVM-LINEAR, SVM-QUADRATIC, ANN, and GPML-SE-ARD is shown in Fig. 105.

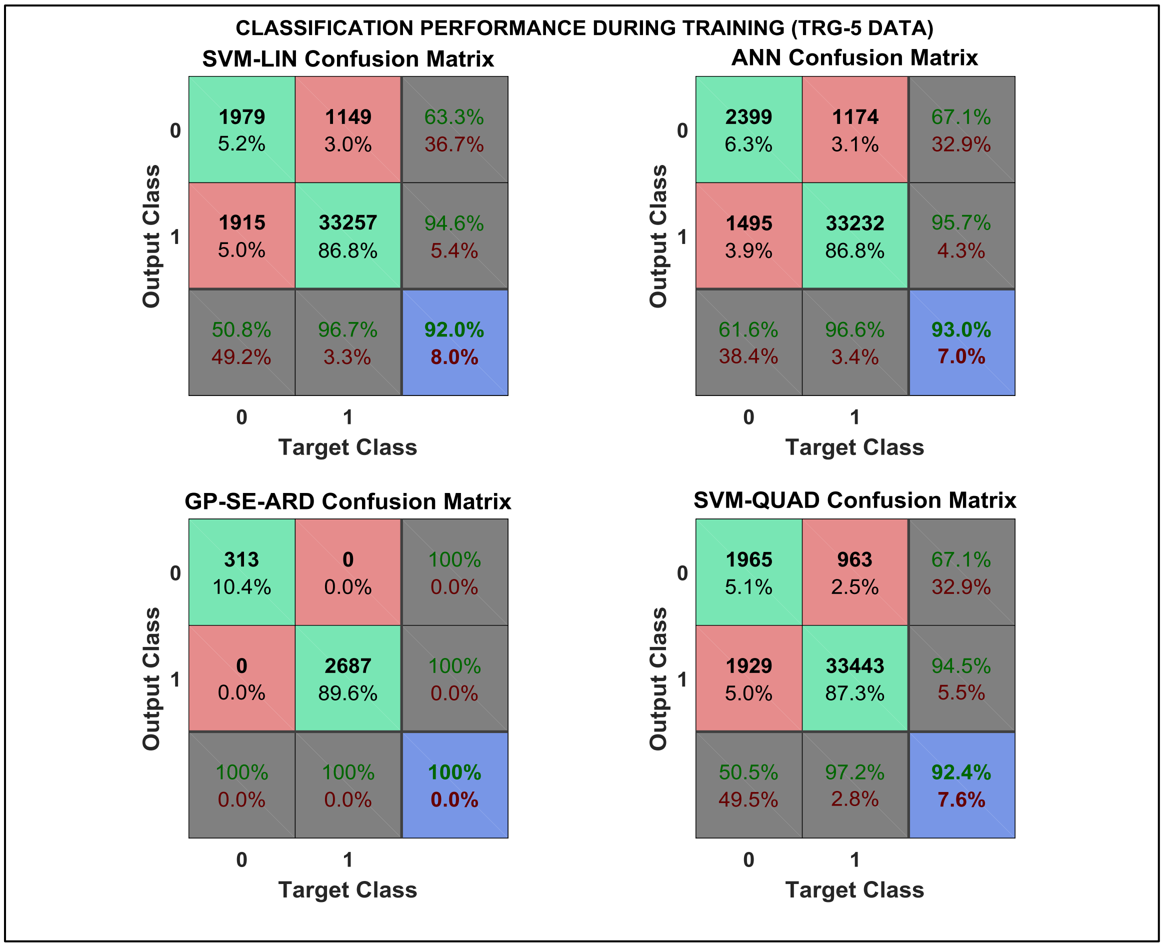


Figure 105: Classification training performance for TRG-5 data.

As shown by the confusion matrix, the GP-SE-ARD classifier performed best with an overall training accuracy of 100%, with the ANN and SVM-QUADRATIC models performing equally well with an overall training accuracy of 93% and 92.4%, respectively. The fourth-best classifier is the SVM-LINEAR model and an overall training accuracy of 92%.

The second statistical classifier description, based on the confusion matrix, is the ROC analysis. The ROC curve for all four classifiers is shown in Fig. 106**.**

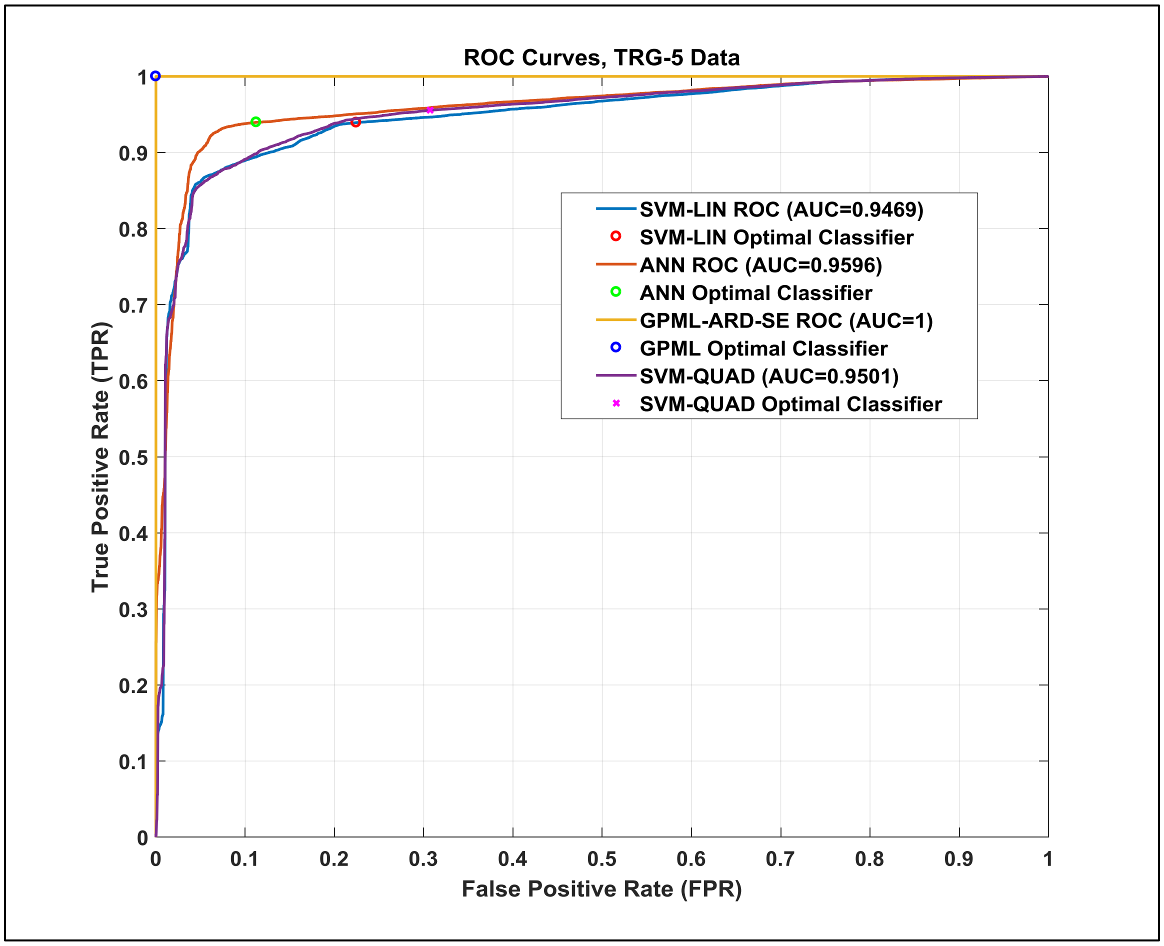


Figure 106: ROC analysis of top four performing classifiers during test (TRG-5 data).

The ROC curve in Fig. 106 shows a test AUC for the SVM-LINEAR, SVM-QUADRATIC, ANN, and GP-SE-ARD of 0.9469, 0.9501, 0.9596, and 1 respectively, all of which indicate excellent, almost perfect classifier performance. For the GP training set that contains 3,000 data points, the GP-SE-ARD classifier performance was perfect. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. The SVM classifier outputs the best guess label 0 or 1 (no upset/upset).

Classifier Performance During Testing – TST-5 Data

To build the performance matrix shown in Fig. 108, the probabilistic output of the ANN and the GP needs to be converted to a binary (upset/no upset) output. To do this, a decision boundary (threshold) needs to be determined above which the probability of upset is set to 1 and below which the probability of upset is set to 0 without compromising the accuracy of the machine. Fig. 107 shows a plot of prediction accuracy vs. threshold for TST-4 data and it indicates that reasonable threshold choice is 0.5, without compromising prediction accuracy.

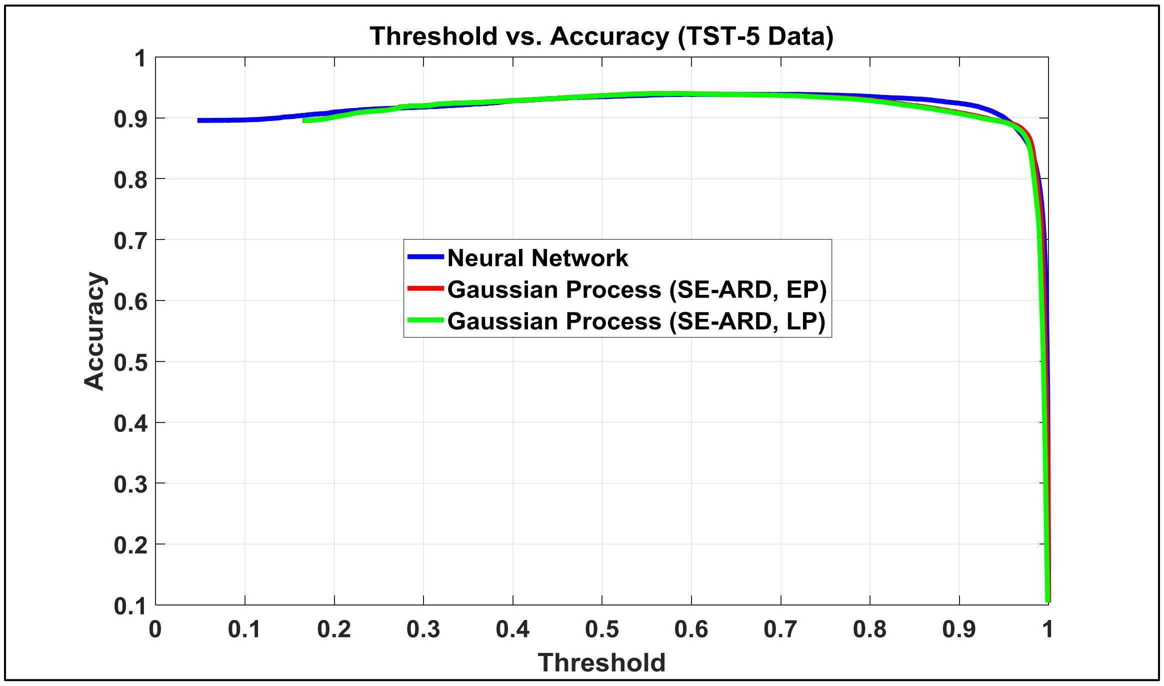


Figure 107: Threshold vs. accuracy for the TST-5 dataset.

The TST-5 test case contains 38,300 test points that will be used to assess the classifier generalization performance. As shown in Fig. 108, on the classification confusion matrix for the TST-5 test case, the GP-SE-ARD and the ANN classifier performed best, with an overall accuracy of 93.7% and 93.5%, respectively, followed by the SVM-QUAD and SVM-LIN classifiers with accuracies of 92.5% and 92 %, respectively. The test performance of all classifiers was slightly worse than performance during training.

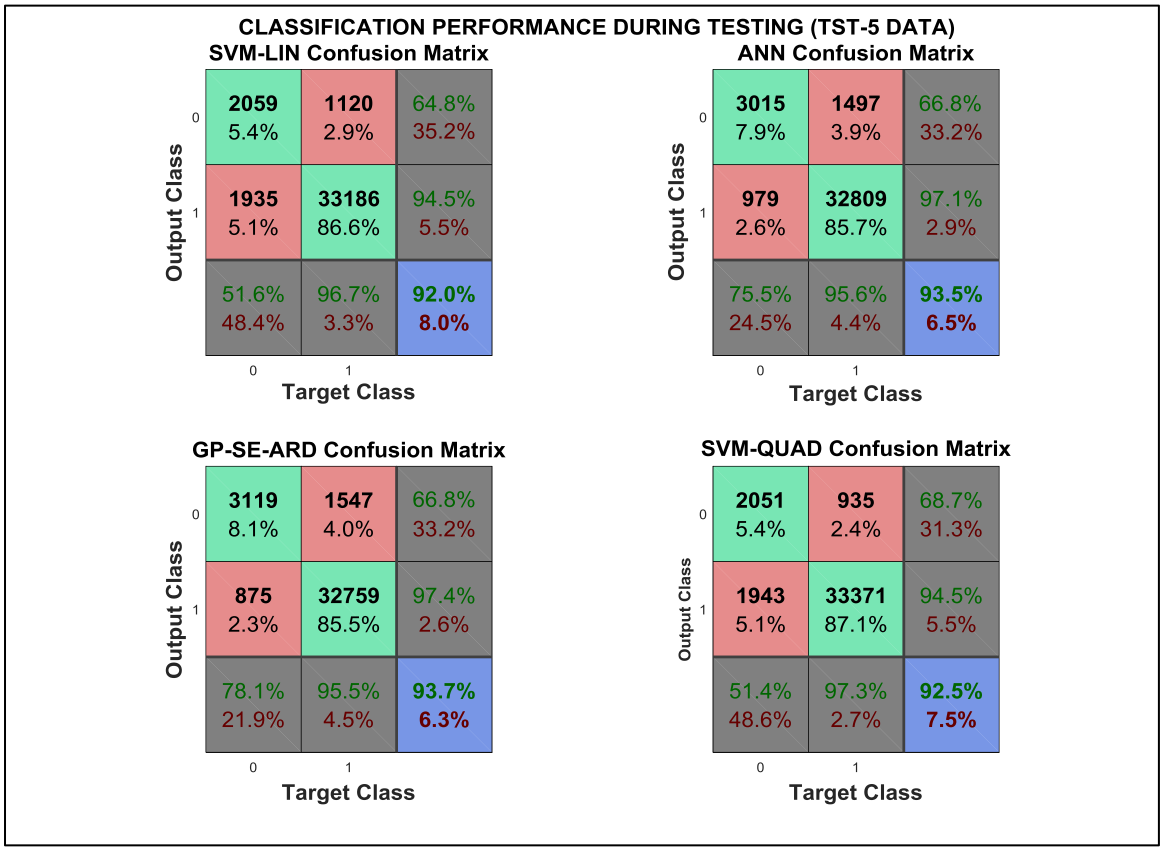


Figure 108: Classifier performance during testing (TST-5 data set).

The ROC curve in Fig. 109 shows a test AUC for the SVM-RBF, SVM-CUBIC, ANN, and GP-SE-ARD of 0.9099, 0.9098, 0.9085, and 0.9073, all of which are in excellent agreements with the respective training AUCs. There is a small miscorrelation between AUC and classifier performance in the confusion matrix because the probabilistic outputs of the ANN and GP are thresholded at 0.5 to obtain a binary classification output. Otherwise, both the ANN and GP output a probability of upset.

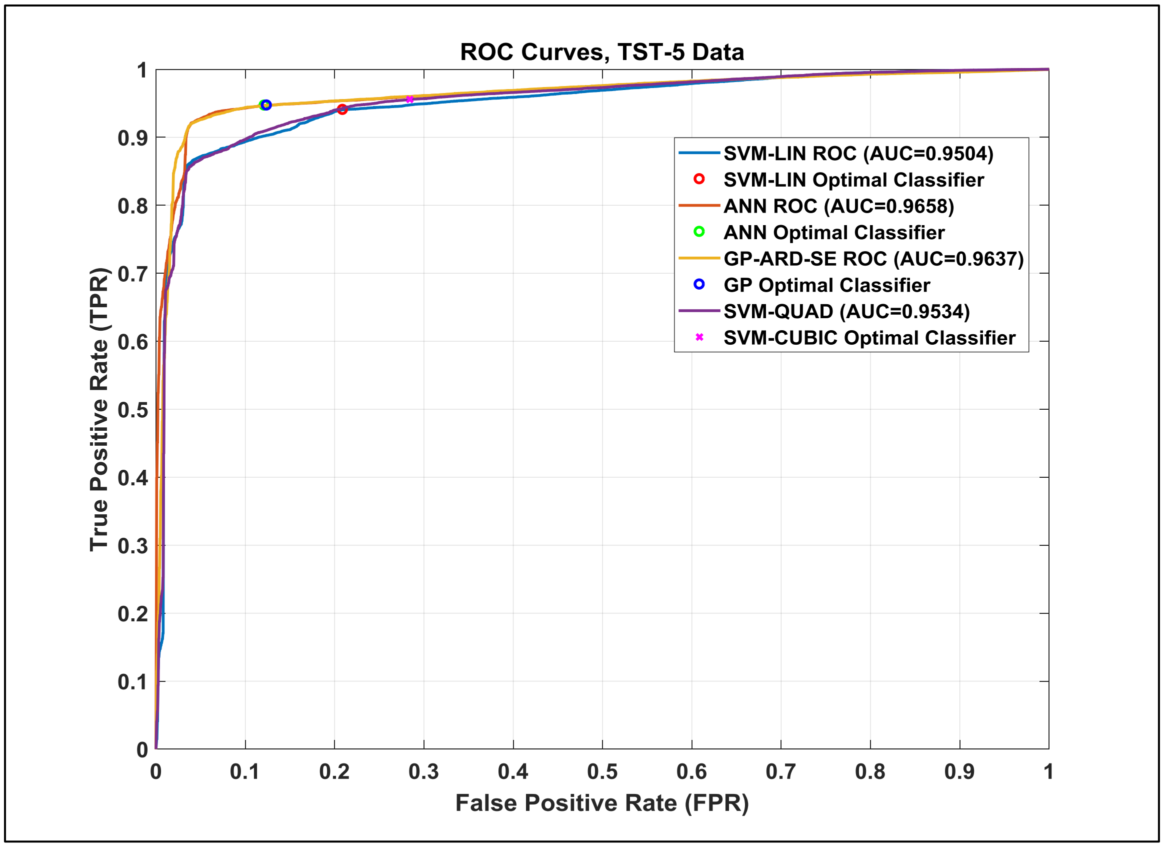


Figure 109: ROC analysis of top four performing classifiers during test (TST-5 data set).

The PoE plot illustrates how the excellent accuracy during training and generalization translates to the classifiers' ability to approximate the PoE as a function of RF pulse injection location for any given injected RF power level, frequency, and pulse width. Just as in the previous two data sets, the PoE plot for the TST-5 case in Fig. 110 is generated by computing the PoE for each position in time for each rising edge of the clock window, each of which contains 100 predicted and 100 measured data points, via the following formula

$\text{Probability of Effect (PoE)} = {}^{\text{Number of Upsets}}/_{\text{Number of Experiments}}$. The SVM-RBF classifier approximates the PoE curve for each clock edge extremely well. As was the case in previous test cases, most of the misclassification contained in the error rates stem from upset misclassifications when the RF pulse is injected during the clock low period of the counting operation. An attempt will be made to improve the classifier performance for the clock low period in the discussion section that following the analysis. Fig. 110 shows how the excellent training and test performance translates to the approximation of the PoE plot as a function of injection location for a given injected RF power level, pulse width, and frequency.

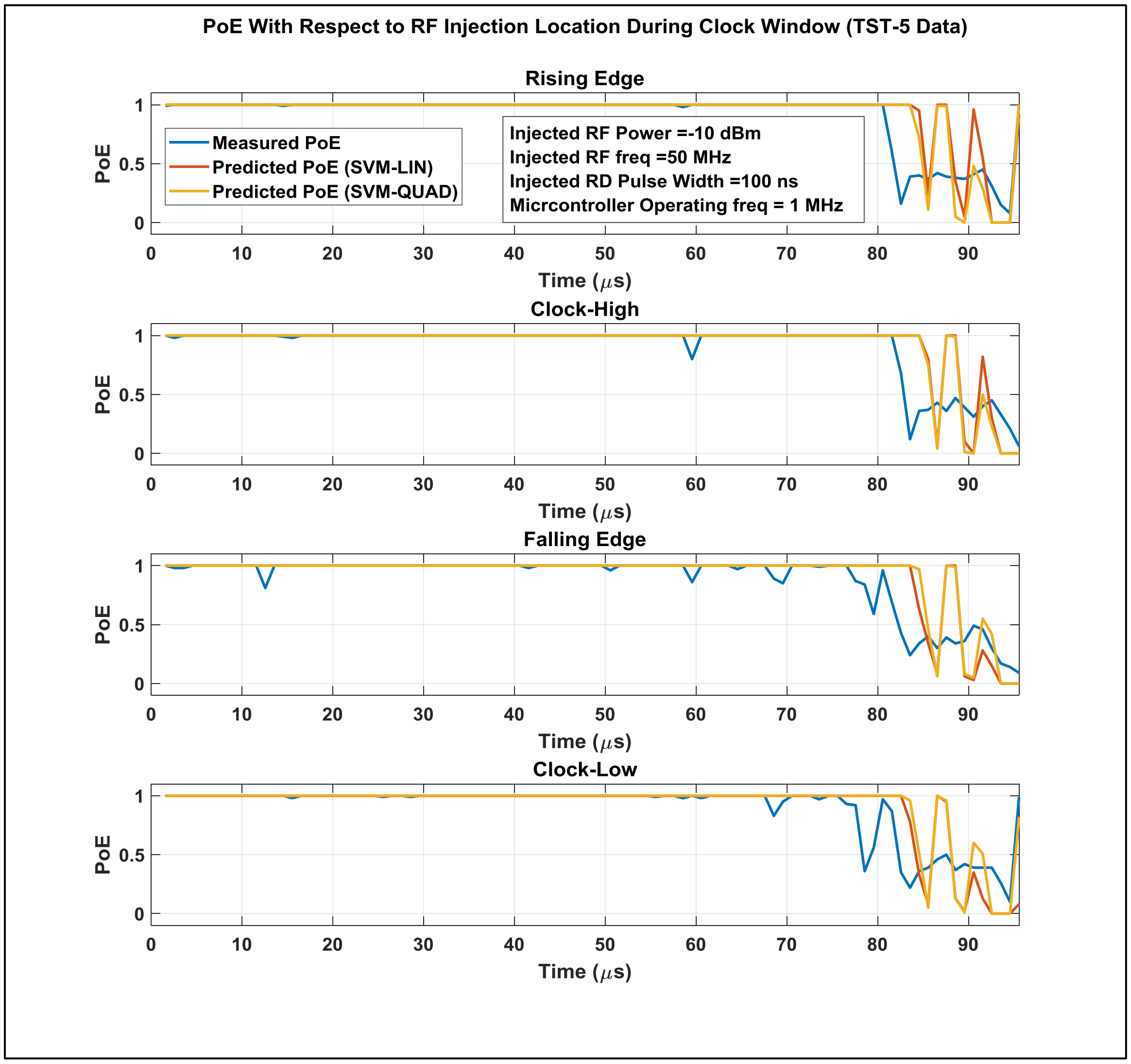


Figure 110: PoE vs. injection location for top two performing SVM classifiers (TST-5 data set).

To reduce clutter, the PoE plot for the TST-5 data set has been split to show performance for the top two SVM classifiers in Fig. 110**,** and the performance of the ANN and GP-SE-ARD classifiers in Fig. 111.

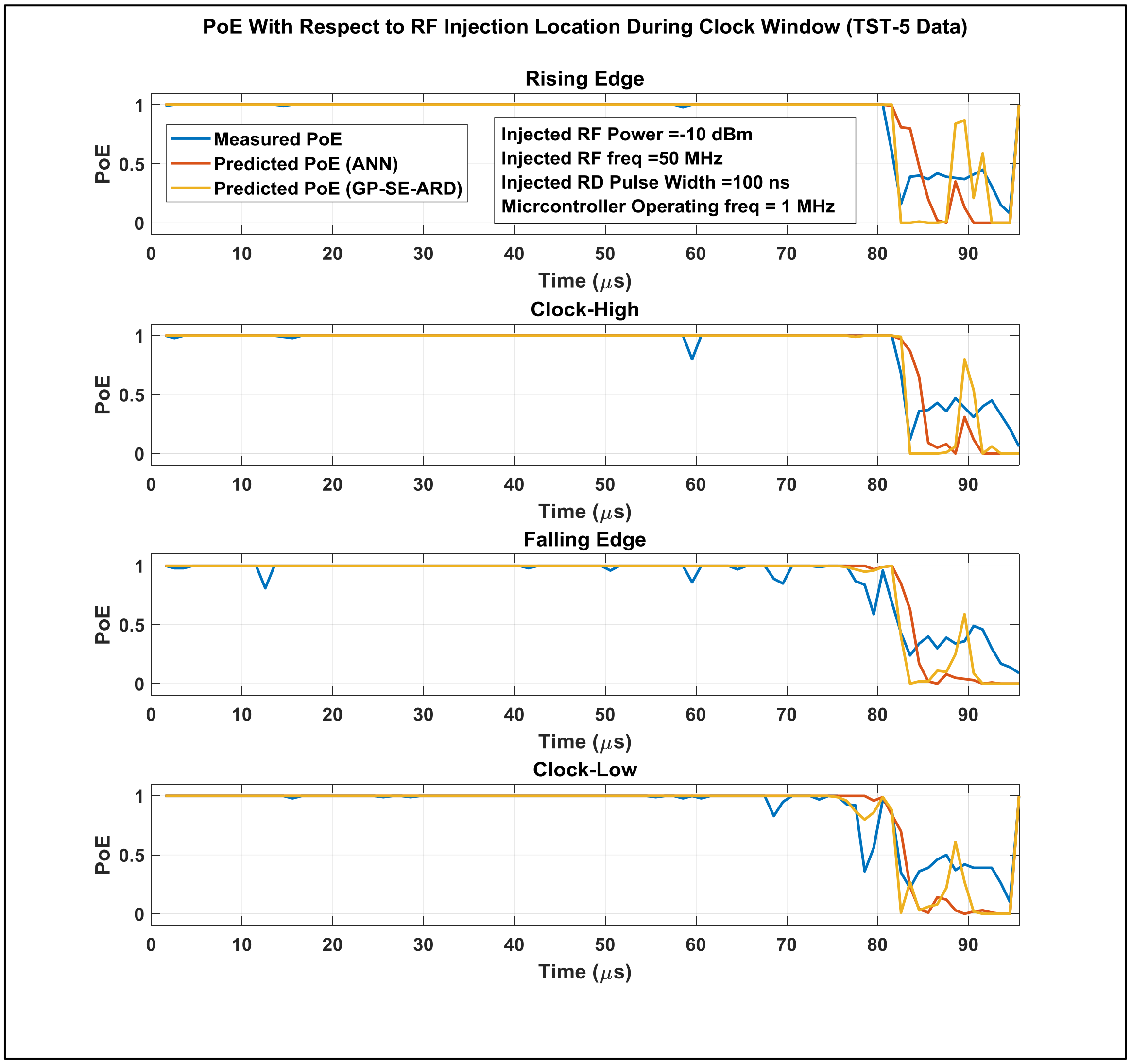


Figure 111: PoE vs. injection location for ANN and GP-SE-ARD classifiers (TST-5 data set).

To summarize the TE-5 test case, all three machine learning methods (SVM, ANN, and GPML) performed very well on TE-5 data, which consisted of 38,300 training points (TRG-5) and 38,300 test points (TST-5). The average training error (TRG-5 data) across all three methods is 0.0712 (7.12%), meaning that they were almost 92.88%

accurate during training, on average, with the actual training accuracy ranging from 89.6% for the SVM-LIN classifier to 100% for the GP-SE-ARD classifier. Performance on the test data (TST-5) was also excellent. The average error across all classifiers for the test data is 0.0818 (8.18%), which is higher than the average training error of 7.12%. The best overall performing machine for TE-5 data is the GP with the squared exponential covariance function that implements automatic relevance determination (GP-SE-ARD). The measurement of training and prediction times is an objective of the research. The single criterion for performance assessment is prediction accuracy. Classifier performance for the TE-5 dataset is summarized in Table 16. The overall measured PoE for the TE-5 test case is 0.8957, with the machine-approximated values being 0.9170, 0.8822, and 0.8782 for the SVM-LINEAR, ANN, and GP-SE-ARD respectively.

Table 16: Classifier performance summary for the TE-5 data set.

TE-5 Data (TRG-5(N)=38,300 / TST-5(N)=38,300, TOTAL(N)=76,600) Injected RF Frequency $(f) = 50$ MHz Injected RF Power $(P) = -10$ dBm Injected RF Pulse Width $(PW) = 100$ ns		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel $C = 3.7666, \quad \sigma = 0.0666$ Bayes Optimization Est. Error: 0.0792	0.0802	0.080
SVM – Quadratic Kernel $C = 9.3326e+04, \quad \sigma = 57.0923$ Bayes Optimization Est. Error: 0.0760	0.0758	0.0750
SVM –Cubic Kernel $C = 1.2546e-04, \quad \sigma = 486.4421$ Bayes Optimization Est. Error: 0.1017	0.1017	0.1040
SVM – Radial Basis Function (RBF) Kernel $C = 7.9906e-05, \quad \sigma = 3.4674$ Bayes Optimization Est. Error: 0.1017	0.1017	0.1040
Artificial Neural Network (ANN)	0.068	0.065
GPs with Expectation Propagation Squared-Exponential Covariance Function w/ARD N=3,000, T=38,300	0	0.063
Machine Learning Method	**Overall PoE (TST-5)**	
	Measured	**Predicted**
SVM - LIN	0.8957	0.9170
ANN		0.8822
GP – SE-ARD (EP)		0.8782

Chapter 5-6: Discussion of Results

While all three machine learning techniques produced excellent results in predicting the probability of upset for the AT89LP2052 MCU, there are several instances in which the predicted PoE curve did not quite match the measure PoE curve during certain clock periods. For the TST-1 data set, the classifier performance for the clock-low cycle was suboptimal, while for the TST-2 and TST-3 data sets, all classifiers could use improvement on the rising-edge, clock high, and clock-low clock cycles. For the TST-5 data set, all classifiers could benefit from modified post-processing to achieve better accuracy starting with the 80 μs point in the clock window.

There are two plausible explanations as to why the classifier performance is not optimal in the cases mentioned above. First, the probabilistic outputs of ANNs and GPs are put through a threshold with a decision boundary at 0.5 that causes lower probability events to be rounded down to zero. While this is a rudimentary boundary, it works quite well for all test cases considered. An improvement in PoE plots between predicted and measured PoEs could be enhanced by not converting the probability outputs of ANNs and GPs to a binary classification value. Second, in the case of SVMs, instead of a probabilistic output, the classifier produces a hard-labeling $\hat{y}(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x}))$, leaving out the measure of confidence in our prediction. Unfortunately, SVMs do not lend themselves to a convenient probabilistic interpretation. One heuristic approach is to interpret $f(x)$ as the log-odds ratio $\log\frac{p(y=1|x)}{p(y=0|x)}$ and convert the output of the SVM to a probability using

$$p(y = 1|, \boldsymbol{x}, \boldsymbol{\theta}) = \sigma(af(\boldsymbol{x}) + b), \tag{121}$$

where a and b can be estimated by maximum likelihood on a separate validation set. However, the resulting probabilities are not well-calibrated, because there is nothing in the SVM training procedure that justifies interpreting $f(x)$ as a log-odds ratio. Hence, additional analysis involving the probabilistic output of $p(y = 1|\boldsymbol{x})$ is done with ANNs and GPs only [12].

For the TST-1 data case, the “non-thresholded” PoE curves are shown in Fig. 112. Averaging the probabilities for each point in time as opposed to thresholding them at 0.5 to produce a binary output produces a much more refined PoE plot that closely matches the measured PoE curve.

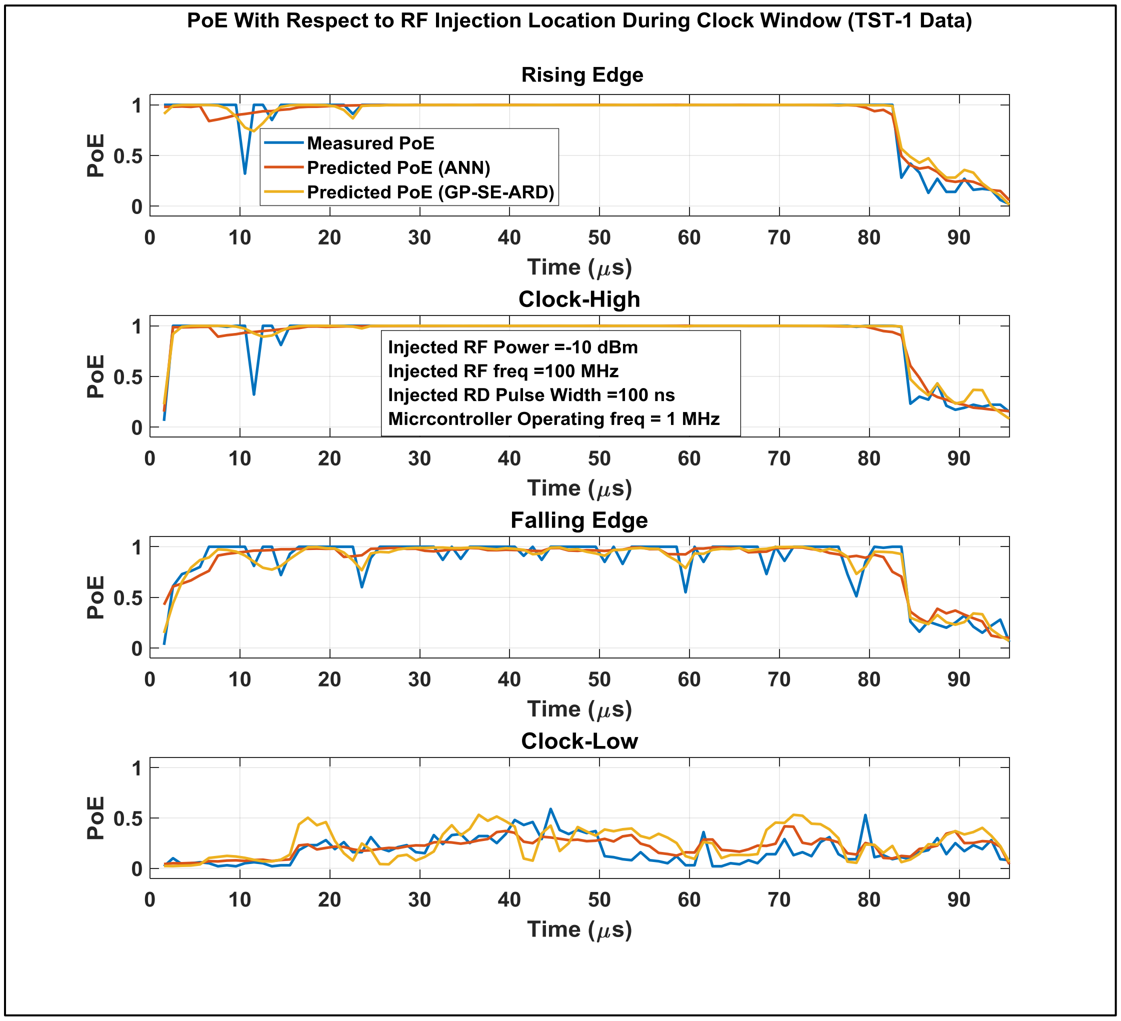


Figure 112: Non-thresholded ANN/GP-SE-ARD classifier performance (TST-1 data).

Similarly, for the TST-2 data, the non-thresholded PoE curves are shown in Fig. 113. Averaging the probabilities for each point in time as opposed to thresholding them at 0.5 to produce a binary output produces a much more refined PoE plot that closely matches the measured PoE curve.

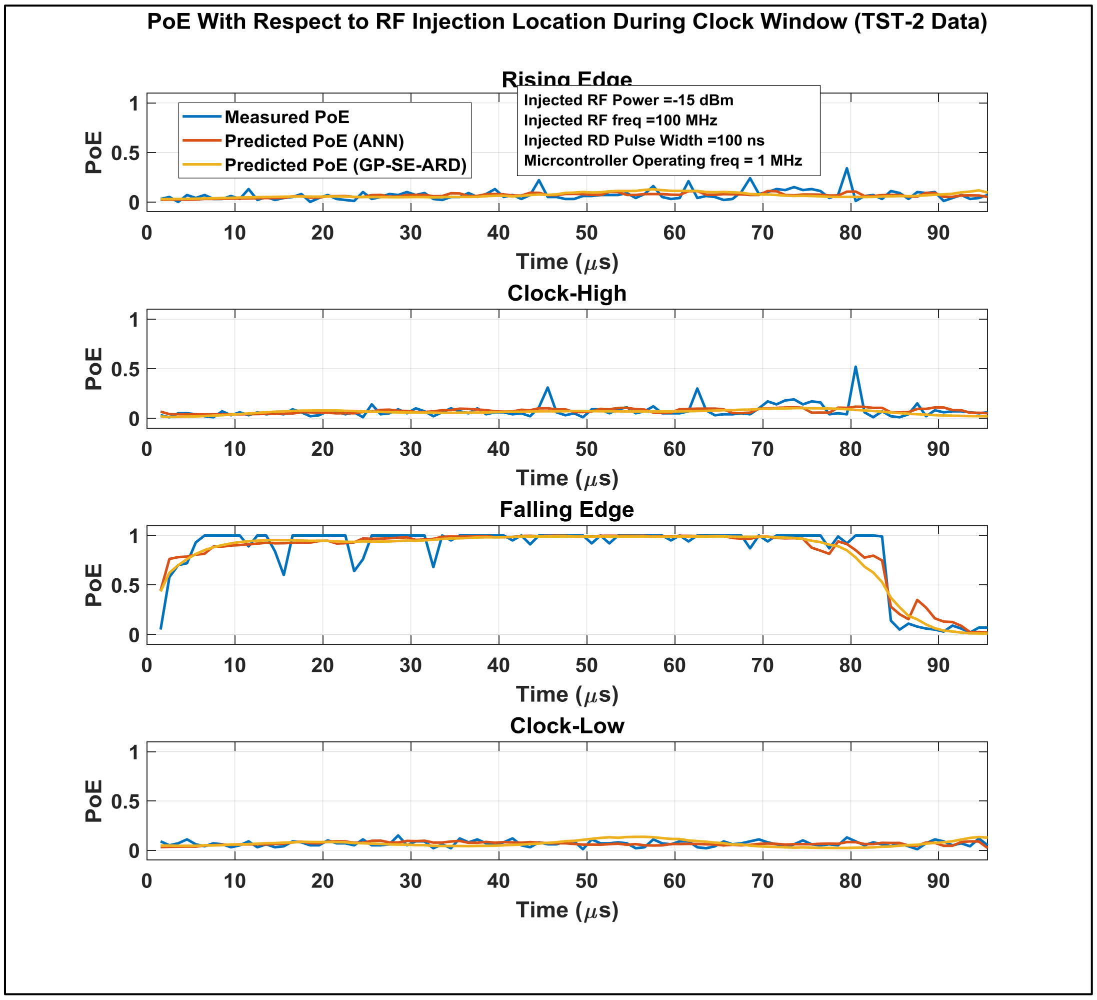


Figure 113: Non-thresholded ANN/GP-SE-ARD classifier performance (TST-2 data).

For the TST-3 data case, the non-thresholded PoE curves are shown in Fig. 114. Indeed, averaging the probabilities for each point in time produces a much more refined PoE plot.

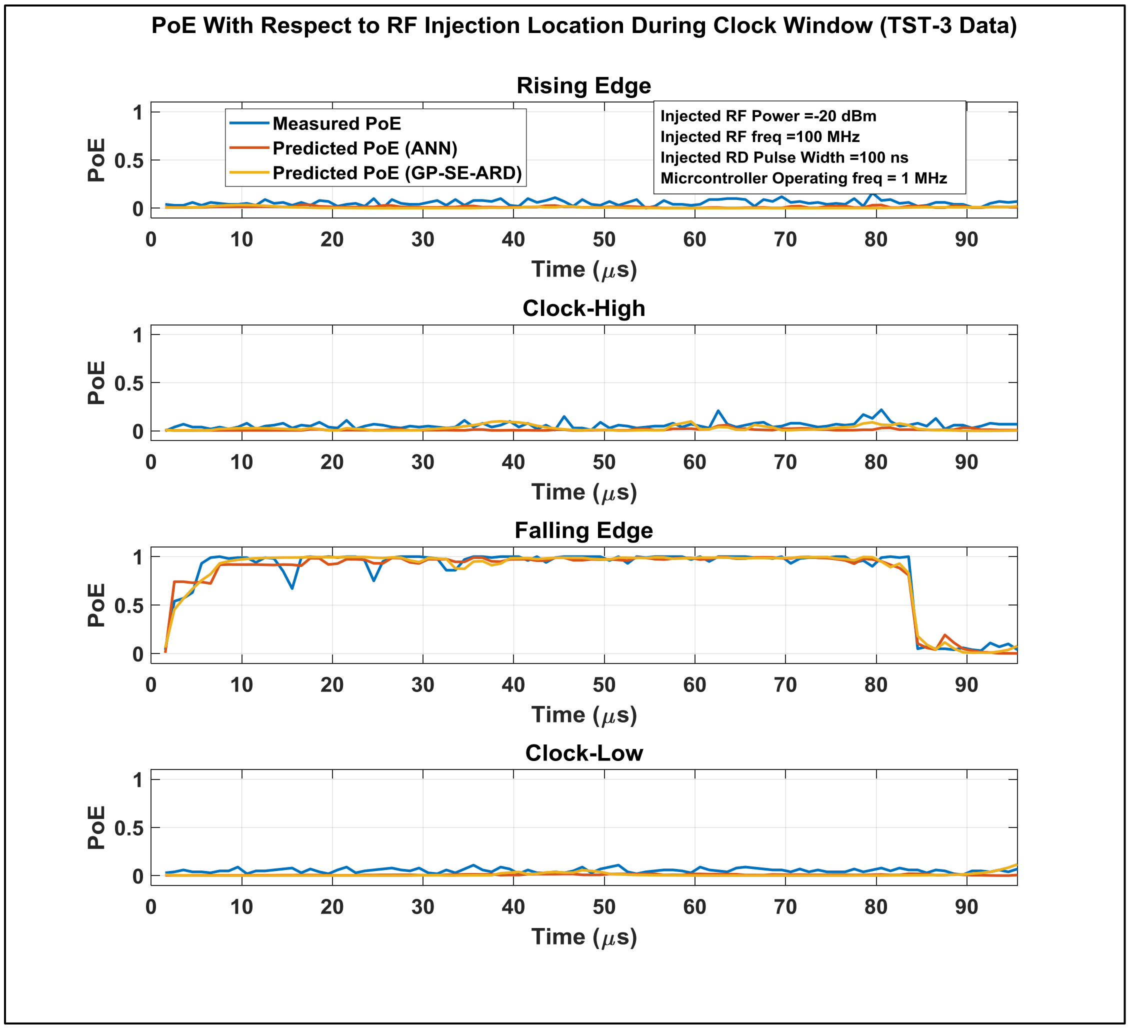


Figure 114: Non-thresholded ANN/GP-SE-ARD classifier performance (TST-3 data).

For the TST-4 data case, the non-thresholded PoE curves are shown in Fig. 115. As before, averaging the probabilities of upset for each point in time (100 measurements) produces a much more refined PoE plot.

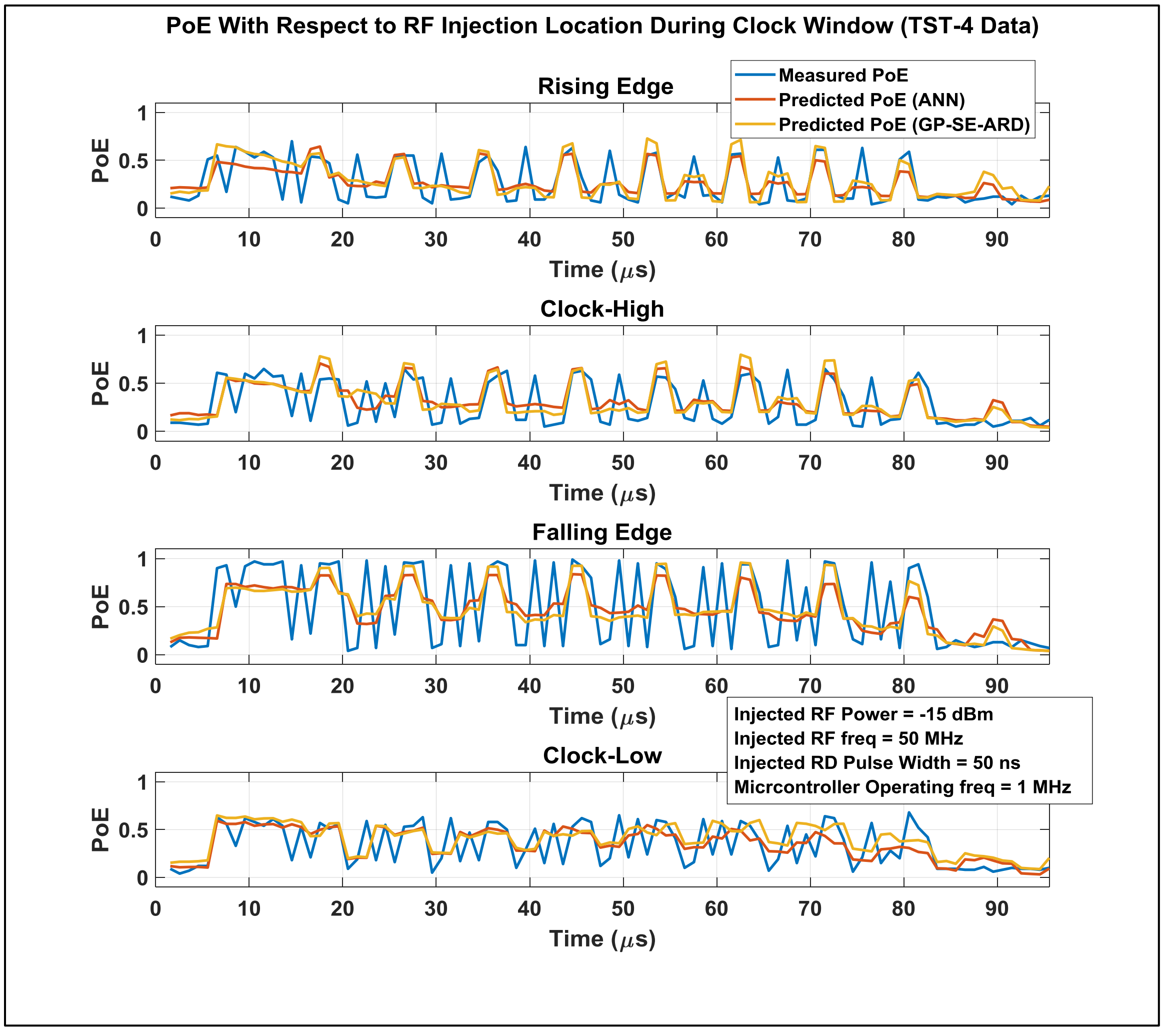


Figure 115: Non-thresholded ANN/GP-SE-ARD classifier performance (TST-4 data).

For the TST-5 data case, the non-thresholded PoE curves are shown in Fig. 116. As before, averaging the probabilities of upset for each point in time (100 measurements) in time produces a much more refined PoE plot.

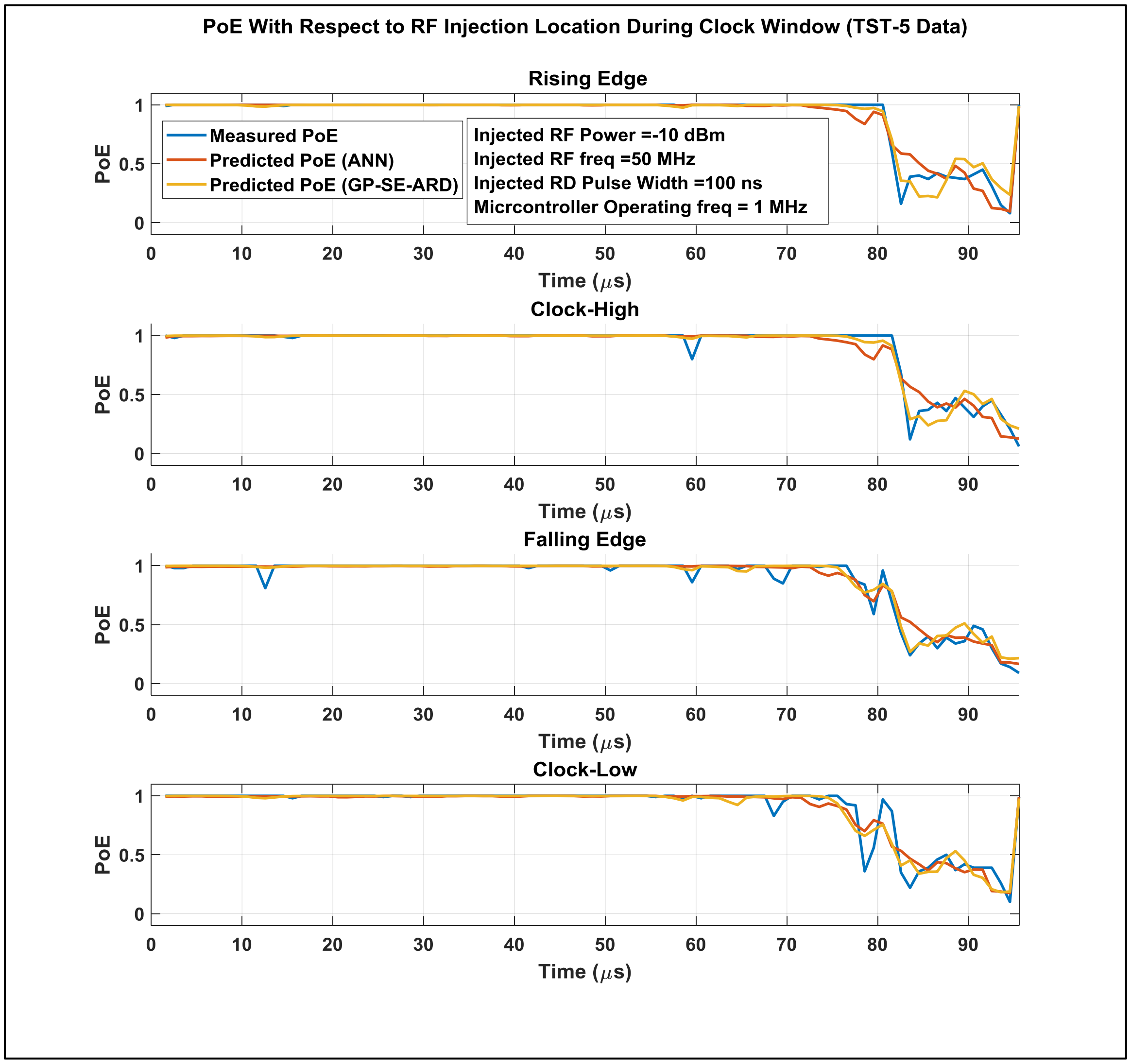


Figure 116: Non-thresholded ANN/GP-SE-ARD classifier performance (TST-5 data).

While overall accuracy remains excellent when the probabilities are put through a threshold to create a binary output, the predicted PoE plots such as the one in Fig. 116 are much more refined when the machine produces a probabilistic output. ANN and GP models naturally produce a probabilistic output, which is why those two methods are featured in the non-thresholded PoE discussion. Standard SVMs, on the other hand, do not provide such probabilities. There are methods that enable probabilistic output for SVMs, such as Platt Scaling [62] and Isotonic Regression [63] that produce results that are competitive with probabilistic methods, but their application to this research is out of scope.

Chapter 6: Summary and Conclusion

Industrialized nations today depend on the fast flow of information as well as information processing systems that make that flow possible. A vast majority of today's information processing systems and infrastructure is not hardened against EMP-producing weapon technology [1]. In addition to unintentional and rudimentary sources of EM radiation that can adversely affect critical electronic infrastructure, advanced militaries are experimenting with the use of HPM weapons systems. The level of technological sophistication that is required to cause electronic system malfunction is not only available to modern militaries but to non-state actors who were previously considered technologically unsophisticated. Potential EMP weapon/device targets range from small systems such as automobiles to highly complex systems such as aircraft, energy and power delivery infrastructure, as well as systems used by the banking and financial sectors [1]. Indeed, there have been several instances of HPEM weapons use against civilian and military infrastructure [2]. Understanding how IEMI affects digital electronics by means of predictive model development is the first important step to developing effective countermeasures.

To simplify system analysis and the initial development of a predictive upset model for electronic systems due to IEMI, the problem that this research proposes to tackle is to construct a computational model for microcontroller upset (MCU) prediction using machine learning tools. The MCU problem presents an ideal choice because it is a complete computer on a single chip representing an intermediate level of system complexity between an individual CMOS device and a complete digital system [7]. Machine learning tools are well-suited to analyze data that describes MCU events because of the inherent probabilistic nature of the problem and the necessity to predict upset events based on previously available data. The statistical nature of the problem, ever-decreasing feature size of ICs, and ever-increasing frequency of operation, call for data-driven models that do not rely on specific technologies and upset mechanisms to effectively predict upset scenarios. A properly designed input-output parameter space together with powerful machine learning techniques have the potential to provide insights into the effects problem that have not been available to date.

During the course of this research, several different methods have been utilized for to model and predict MCU upset, two of which are kernel machines (SVMs, GPs) and ANNs, which are not kernel-based. All kernel methods require tuning of kernel parameters. Since GPs are based on Gaussian priors, the marginal likelihood is maximized using efficient gradient-based optimizers whereas SVM parameters (C and σ) are determined by minimizing the 10-fold cross-validation loss using Bayesian optimization. ANNs use interconnect weights that are determined by data via a scaled-conjugate-gradient method with error backpropagation. SVMs are sparse methods, meaning that they only use a subset of the training examples, while GPs and ANNs are not sparse – they use all the training examples. The main advantage of sparse methods is that prediction at test time is usually faster, and sometimes they produce results with better accuracy, however in the MCU upset scenario, SVMs did not produce significantly improved accuracy when compared to GPs. Additionally, GPs and ANNs produce a probabilistic output of the form $P(y|\boldsymbol{x})$, whereas SVMs produce a confidence value that can be converted to probability, but these probabilities are not well-calibrated and hence may produce suboptimal results.

In MCU upset experiments, all three methods had comparable accuracy when averaged over a range of test cases, provided the regularization constants are chosen appropriately. Given that the performance of all three models is roughly the same, a criterion for additional model comparison is computational performance. Because GPs do not exploit sparsity, they are generally the slowest and computationally most expensive scaling with $\mathcal{O}(n^3)$, while SVM also scale with $\mathcal{O}(n^3)$ during training, unless a linear kernel is used, in which case we only need $\mathcal{O}(n^2)$ computational time. Therefore, suitability of each model will depend on additional performance criteria, the detailed study of which is not within the scope of this research.

GPs can be used to specify prior distributions for Bayesian inference. On the one hand, in the case of regression with Gaussian noise, inference can be computed in closed form, because the posterior is also a GP. On the other hand, for non-Gaussian likelihoods, such as e.g., in binary classification, exact inference is analytically intractable, and we must resort to approximations, such as Laplace approximation [64], expectation propagation (EP) [48] and Kullback-Leibler divergence (KL) minimization [65] comprising of Variational Bounding (VB) as a special case [66]. Another approach is based on a factorial approximation, rather than a Gaussian approximation [67].

The MCU upset data was collected and analyzed using various MATLAB toolboxes [68] and the latest release of GPML [69] software. For the five datasets composed of 191,500 data points (383,000 if training data is included), the average training accuracy was 88.25%, 90.48%, and 91.88%, for the SVM, ANN, and GP approach respectively. The average test accuracy was determined to be 86.91%, 89.2%, and 88.96 for the SVMs, ANNs, and GPs respectively. If test prediction accuracy is our primary criterion for model selection, then the ANN approach is preferred, even though all three methods produced excellent results in terms of prediction accuracy. Model complexity and computational complexity are other possible performance criteria, according to which we can evaluate model suitability. The results are summarized in Table 17.

Table 17: Machine prediction of MCU upset - summary of results.

TE-ALL Data (TRG (N)=191,500 / TST (N)=191,500, TOTAL(N)=383,000) Injected RF Frequency $(f) = 50, 100\ MHz$, Injected RF Power $(P) = -10, -15, -20\ dBm$, Injected RF Pulse Width $(PW) = 50, 100\ ns$		
Machine Learning Technique	**Error Rates**	
	Training	**Test**
SVM – Linear Kernel	0.1544	0.1837
SVM – 2nd Degree Polynomial (Quadratic) Kernel	0.1018	0.1146
SVM – 3rd Degree Polynomial (Cubic) Kernel	0.1022	0.1170
SVM – Radial Basis Function (RBF/Gaussian) Kernel	0.0954	0.1086
Artificial Neural Network (ANN)	0.0952	0.1080
GPs with SE-ARD Covariance Function (EP) / likERF	0.0743	0.1104
GPs with SE-ARD Covariance Function (LP) / likERF	**0.0888**	**0.1076**
AVERAGE ERROR	**0.1247**	**0.1485**

Looking at the data, the overall trend is that shorter, lower power, RF pulse widths cause upset less frequently than longer RF signals of higher power. It is also interesting that the overall PoE for the TE-1 dataset ($f = 100$ MHz, $P = -10$ dBm, $PW = 100$ ns) was measured to be approximately 0.70 and predicted to be approximately 0.65, while the overall PoE for the TE-5 dataset ($f = 50$ MHz, $P = -10$ dBm, $PW = 100$ ns) was measured and predicted to be almost 0.90, indicating that injected RF signal frequencies that are closer to the operating frequency of the MCU have a higher likelihood of causing upset. This limited dataset indicates that all variables studied are important in causing MCU upset, and more experiments will need to be performed to fully characterize the effect of each variable on the PoE and the effect of data composition on machine prediction accuracy, before more general conclusions can be made.

As far as machine learning accuracy is concerned, it appears that the machine is more accurate on imbalanced datasets, i.e. if the data is imbalanced toward upset or no upset. The closer the overall PoE is to 0 or 1, the more accurate machine learning prediction becomes, as evidenced by datasets TE-2, TE-3, and TE-5. For example, datasets TE-2 and TE-3 have an overall PoE of about 0.25, and an overall prediction error of 0.20 to 0.25, on average, while dataset TE-5 has an average error of approximately 0.071 with an overall PoE of 0.89. Datasets TE-1 and TE-4 are less imbalanced with overall PoEs of 0.69 and 0.34, respectively. The average error for the less imbalanced datasets TE-1 and TE-4 is 0.11 and 0.26, respectively. More experiments, over a wider range of injected RF frequencies, injected RF power, injections locations, and injected RF pulse width are needed to build more comprehensive models.

Three machine learning methods, SVMs, ANNs, and GPs were applied to a very difficult problem of modeling MCU upset when it is formulated as a binary classification problem with excellent results. None of the three techniques used to analyze MCU upset data is without limitation. While ANN models provide the best accuracy during the test, the method suffers from poor interpretability. SVMs models also produce excellent predictions but do not provide well-calibrated probabilistic output for the same, and the GP method computational complexity scales with $\mathcal{O}(n^3)$, making training and prediction computationally expensive. Deep learning kernel methods have the potential to alleviate the computational expense associated with GPs and SVM training, but they are unlikely to resolve the remaining limitations. There are numerous ways to expand on this promising approach, and they are discussed in the next chapter.

Chapter 7: Future Research

This research assessed the suitability of three popular machine learning methods (SVMs, ANNs, GPs) to model and predict MCU upset with excellent results. There are several recommendations on how to expand on this research: (1) enhance the experiment setup to reduce the noise level present in the system (2) fully explore the available model space by utilizing automated model selection methods as described in [70], [71]; (3) include analysis of two simultaneously injected RF pulses; (4) broaden the definition of upset to discriminate among several different types and repeat the analysis under a multi-class classification paradigm; (5) expand the number of models under consideration to include two-class decision forest, two-class boosted decision tree, two-class logistic regression, among many other binary classification algorithm; (6) explore regression methods to directly estimate the PoE, which may raise the prediction accuracy; (7) explore the use of recently-popular deep learning techniques such as deep kernel methods (SVMs, GPs), convolutional neural networks (CNNs), among many others; (8) utilize ensemble learning methods. Four immediate recommendations are to improve the experiment setup, use automated model selection methods, use deep learning techniques, and use ensemble learning techniques to achieve better scalable results.

The reduction of noise in the experimental setup can be achieved in several ways. This can be achieved by replacing a dated RF sweeper and acquisition oscilloscope. Data acquisition speed can be optimized by revamping SALVO acquisition software to acquire data via ethernet, rather than via USB/GPIB connections. It is estimated that data acquisition time can be reduced by a factor of 10, and with the reduction of noise in the system, better data quality and quantity can improve prediction accuracy.

Malkomes et al. [70] propose an automatic framework for exploring a set of potential models with the goal to select the model that best explains the given data. Even though they focus on GP models, their method can be easily extended to any probabilistic model with a parametric or structured model space. Specifically, they apply a Bayesian optimization method to explore a model space, a process that they demonstrated to be capable of predicting the evidence value of unseen models with sufficient fidelity. In his Ph.D. work, Duvenaud [71] explores the automatic construction of models based on GPs that can capture many types of a statistical structure such as periodicity, changepoints, additivity, and symmetries, the nature of which is encoded through kernel selection. He describes an automated kernel selection and construction process, a task that has traditionally been considered a “black art” performed by data experts.

Deep learning computational models are composed of multiple processing layers that are used to learn a representation of data by using multiple levels of abstraction, often increasing accuracy over “shallow” methods. CNNs are designed to process data in the form of multiple arrays, such as a color image, or really any other data or phenomenon whose characteristics can be described by an array of data – MCU upset data being one of them. CNNs take advantage of the properties of natural signals: local connections, shared weights, pooling, and the use of many layers. Recent CNN architectures have 10-20 layers of rectifier linear units (ReLUs), hundreds of millions of weights, and billions of connections between units. Two years ago, training these neural networks would have taken weeks, advancements in hardware, software, and algorithm parallelization have reduced that time to a few hours. In the field of image recognition, the performance of CNNs is approaching the human level [72]. It would be very interesting to investigate how CNNs would perform on MCU upset data.

The utilization of ensemble learning is another immediate recommendation to expand this research. Ensemble learning is a method for constructing ensembles from libraries of thousands of models [73]. The method utilizes a forward stepwise selection to add to the ensemble models, that maximize performance on a particular set of data. Ensemble selection allows ensembles to be optimized based on performance metrics such as accuracy, cross-entropy, mean precision, or ROC area under the curve (AUC). Caruana et al. used the ensemble model to train 2000 different models for each problem by using SVMs, ANNs, decision trees (DT), boosted decision trees (BST-DT), among others. Some models resulted in an excellent performance, while others did not, and rather than combine good and bad models in an ensemble, a forward stepwise selection from the library of models is used to find a subset of models that, when combined, yields excellent performance. Ensemble selection finds ensembles that outperform the best single models. The most important feature of ensemble selection is that it can optimize performance to any easily-computed performance metric, such as accuracy, cross-entropy, mean-precision, or ROC AUC, almost all of which were utilized during this research. It would be very interesting to examine whether ensemble learning could stitch together several models that will outperform SVMs, ANNs, and GPs on MCU upset data.

Bibliography

[1] J. Benford, J. A. Swegle, and E. Schamiloglu, *High Power Microwaves*, 3rd. Boca Raton, FL: CRC Press, 2016.

[2] W. A. Radasky, C. E. Baum, and M. W. Wik, “Introduction to the Special Issue on High-Power Electromagnetics (HPEM) and Intentional Electromagnetic Interference (IEMI),” *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 3, pp. 314–321, 2004.

[3] D. V. Giri and F. M. Tesche, “Classification of intentional electromagnetic environments (IEME),” *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 3, pp. 322–328, 2004.

[4] “Magnetron – Wikipedia.” [Online]. Available: https://hu.wikipedia.org/wiki/Magnetron. [Accessed: 01-Jun-2017].

[5] I. Radio Research Instrument Co., “Radio Research Instrument Co., Inc.” [Online]. Available: http://pe2bz.philpem.me.uk/Comm01/- - Ion-Photon-RF/- - Radar - - /Prod-100-Userd/radiores.html.

[6] M. Backstrom and K. Lovstrand, “Susceptibility of Electronic Systems to High-Power Microwaves: Summary of Test Experience,” *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 3, pp. 396–403, 2004.

[7] T. Clarke, “Modeling of High Power Electromagnetic Effects on Digital Electronics,” Air Force Office of Scientific Research Annual Report, 2013.

[8] G. Oliveri, P. Rocca, and A. Massa, “SVM for Electromagnetics: State-of-art,

Potentialities, and Trends," in *IEEE Antennas and Propagation Society, AP-S International Symposium (Digest)*, 2012, pp. 7–8.

[9] R. M. Neal, "Bayesian Learning for Neural Networks," Springer Verlag, New York, NY, 1996.

[10] W. S. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[11] F. Rosenblatt, "A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[12] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.

[13] S. Wendsche, R. Vick, and E. Habiger, "Modeling and testing of immunity of computerized equipment to fast electrical transients," *IEEE Trans. Electromagn. Compat.*, vol. 41, no. 4 PART 2, pp. 452–459, 1999.

[14] R. Vick and E. Habiger, "The dependence of the immunity of digital equipment on the hardware and software structure," in *International Symposium on Electromagnetic Compatibility Proceedings*, 1997, pp. 383–386.

[15] H. Wang, C. Dirik, S. V Rodriguez, A. V Gole, and B. Jacob, "Radio Frequency Effects on the Clock Networks of Digital Circuits," in *2004 International Symposium on Electromagnetic Compatibility*, 2004, pp. 93–96.

[16] D. Nitsch and M. Camp, "Susceptibility of Some Electronic Equipment to HPEM Threats," *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 3, pp. 380–389, 2004.

[17] M. Camp and H. Gerth, “Predicting the Breakdown Behavior of Microcontrollers Under EMP/UWB Impact Using a Statistical Analysis,” *IEEE Trans. Electromagn. Compat.*, vol. 46, no. 3, pp. 368–379, 2004.

[18] S. Y. Yuan, “A Microcontrller Conducted EMI Model Building for Software-level Effect,” in *9th International Workshop on Electromagnetic Compatibility of Integrated Circuits*, 2013, pp. 186–189.

[19] S. Caorsi and G. Cevini, “A Neural Network Approach for Electromagnetic Diagnostic Applications,” *PIERS Online*, vol. 2, no. 4, pp. 385–389, 2006.

[20] D. Micu, L. Czumbil, G. Christoforidis, and A. Ceclan, “Layer Recurrent Neural Network Solution for an Electromagnetic Interference Problem,” *IEEE Trans. Magn.*, vol. 47, no. 5, pp. 1410–1413, 2011.

[21] V. Ceperic and A. Baric, “Modelling of Electromagnetic Immunity of Integrated Circuits by Artificial Neural Networks,” in *2009 20th International Zurich Symposium on Electromagnetic Compatibility*, 2009, pp. 373–376.

[22] X. Li, J. Yu, Y. Zhu, Q. Wang, and Y. Li, “Prediction of Electromagnetic Compatibility Problems Based on Artificial Neural Networks,” in *2008 World Automation Congress*, 2008.

[23] I. Chahine, M. Kadi, E. Gaboriaud, A. Louis, and B. Mazari, “Using Neural Networks for Predicting the Integrated Circuits Susceptibility to Conducted Electromagnetic Disturbances,” in *Proceedings of the 18th International Zurich Symposium on Electromagnetic Compatibility, EMC*, 2007, no. C, pp. 13–16.

[24] V. Devabhaktuni, C. F. Bunting, D. Green, D. Kvale, L. Mareddy, and V. Rajamani, “A New ANN-based Modeling Approach for Rapid EMI/EMC Analysis of PCB and Shielding Enclosures,” *IEEE Trans. Electromagn. Compat.*, vol. 55, no. 2, pp. 385–394, 2013.

[25] V. Ceperic, G. Gielen, and A. Baric, “Black-box Modelling of Conducted Electromagnetic Immunity by Support Vector Machines,” in *IEEE International Symposium on Electromagnetic Compatibility*, 2012, no. 1.

[26] V. Ceperic, G. Gielen, and A. Baric, “Black-box modelling of conducted electromagnetic immunity by support vector machines,” *IEEE Int. Symp. Electromagn. Compat.*, no. 1, 2012.

[27] Y. Wu, Z. Tang, B. Zhang, and Y. Xu, “Using Support Vector Machine Regression for Measuring Electromagnetic Parameters of Magnetic Materials,” in *International Symposium on Microwave, Antenna, Propagation, and EMC Technologies for Wireless Communications*, 2007, pp. 1020–1022.

[28] Y. Wu, Z. Tang, B. Zhang, and Y. Xu, “Using Support Vector Machine Regression for Measuring,” pp. 1020–1022, 2007.

[29] Y. S. Abu-Mostafa, M.-I. Malik, and H.-T. Lin, *Learning From Data: A Short Course*. United States of America: AML Book, 2012.

[30] M. Martínez-Ramón and C. Christodoulou, *Support Vector Machines for Antenna Array Processing and Electromagnetics*, 1st Editio., vol. 1, no. 1. United States of America: Morgan & Claypool Publishers, 2006.

[31] N. Christianini and J. Shawe-Taylor, *Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge: University Press, 2003.

[32] M. Herst, T. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support Vector Machines,” *Wikipedia*, 1998. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed: 11-Oct-2016].

[33] MathWorks, “Support Vector Machines for Binary Classification,” 2015. [Online]. Available: http://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html. [Accessed: 29-Mar-2017].

[34] R. ter Borg and F. ter Borg-Brinkman, “Mercer Kernel.” [Online]. Available: https://www.terborg.net/research/kml/reference/mercer_kernel.html. [Accessed: 11-Nov-2016].

[35] B. Scholkopf and J. Smola, Alexander, *Learning with Kernels*. MIT Press, 2002.

[36] J. Friedman and T. Hastie, *The Elements of Statistical Learning*. New York, NY: Springer Verlag, 2008.

[37] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” 2012.

[38] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesus, *Neural Network Design*, 2nd ed. San Bernadino, CA: Martin Hagan, 2015.

[39] B. Widrow and M. Hoff, “Adaptive switching circuits.,” *1960 IRE WESCON Conv. Rec.*, no. 4, pp. 96–104, 1960.

[40] D. E. Rumelhart and J. L. McCellland, *Parallel Distributed Processing*.

Cambridge, MA: MIT Press, 1986.

[41] M. F. Møller, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning,” *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[42] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.

[43] K. Levenberg, “A Method for the solution of certain non-linear probles in least squares.,” *Quart. Appl. Math*, vol. 11, no. 2, pp. 164–168, 1944.

[44] D. J. C. MacKay, “Introduction to Gaussian Processes.”

[45] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.

[46] C. Sammut and G. Webb, “Gaussian Processes,” *Encyclopedia of Machine Learning*. Springer Verlag, New York, 2010.

[47] M. Kuss and C. E. Rasmussen, “Assessing Approximate Inference for Binary Gaussian Process Classification,” *J. Mach. Learn. Res.*, vol. 6, pp. 1679–1704, 2005.

[48] T. P. Minka, “Expectation Propagation for Approximate Bayesian Inference,” *Uncertain. Artif. Intell.*, vol. 17, no. 2, pp. 362–369, 2001.

[49] L. Stein, Michael, *Interpolation of Spatial Data: Some Theory for Kriging*. 1999.

[50] B. X. Lin, G. Wahba, D. Xiang, F. Gao, R. Klein, and B. Klein, “Smoothing Spline Anova Models for Large Data Sets,” *Ann. Stat.*, vol. 28, no. 6, pp. 1570–1600, 2000.

[51] J. Zhu and T. Hastie, "Kernel Logistic Regression and the Import Vector Machine," *J. Comput. Graph. Stat.*, vol. 14, no. 1, pp. 185–205, 2005.

[52] N. Lawrence, M. Seeger, and R. Herbrich, "Fast Sparse Gaussian Process Methods: The Informative Vector Machine," *Adv. Neural Inf. Process. Syst. 15*, pp. 625–632, 2003.

[53] L. Csató and M. Opper, "Sparse On-line Gaussian Processes.," *Neural Comput.*, vol. 14, pp. 641–668, 2002.

[54] V. Tresp, "A Bayesian Committee Machine," *Neural Comput.*, vol. 2741, no. 12, pp. 2719–2741, 2000.

[55] Stanford Research, "Function Generator - DS345." [Online]. Available: http://www.thinksrs.com/products/DS345.htm.

[56] ATMEL Corporation, "AT89LP4052." ATMEL Corporation.

[57] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Data Mining Researchers ROC Graphs : Notes and Practical Considerations for Data Mining Researchers," 2003.

[58] Mathworks, "Detector Performance Analysis Using ROC Curves - MATLAB & Simulink Example," *Www.Mathworks.Com*. [Online]. Available: http://www.mathworks.com/help/phased/examples/detector-performance-analysis-using-roc-curves.html. [Accessed: 14-Apr-2017].

[59] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[60] Wikimedia Commons, “ROC space,” 2009. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/3/36/ROC_space-2.png. [Accessed: 14-Apr-2017].

[61] F. Gorunescu, *Data Mining: Concepts and Techniques*. Berlin: Springer Verlag, 2011.

[62] J. Platt, “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods,” *Adv. Large Margin Classif.*, vol. 10, no. 3, pp. 61–74, 1999.

[63] B. Zadrozny and C. Elkan, “Transforming Classifier Scores into Accurate Multiclass Probability Estimates,” in *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, 2002, p. 694.

[64] C. K. I. Williams and D. Barber, “Bayesian Classification with Gaussian Processes,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 12, pp. 1342–1351, 1998.

[65] M. Opper and C. Archambeau, “The Variational Gaussian Approximation Revisited,” *Neural Comput.*, vol. 21, no. 3, pp. 786–792, 2009.

[66] M. N. Gibbs and D. J. C. MacKay, “Variational Gaussian Process Classifiers,” *IEEE Trans. Neural Networks*, vol. 11, no. 6, pp. 1458–1464, 2000.

[67] L. Csató, E. Fokoue, M. Opper, B. Schottky, and O. Winther, “Efficient approaches to Gaussian process classification,” *Adv. Neural Inf. Process. Syst. 12*, vol. 12, no. x, pp. 251–257, 2000.

[68] Mathworks, “MATLAB.” Mathworks, Inc., Natick, MA, 2017.

[69] C. E. Rasmussen and C. K. I. Williams, “Gaussian Processes for Machine Learning Software.” 2016.

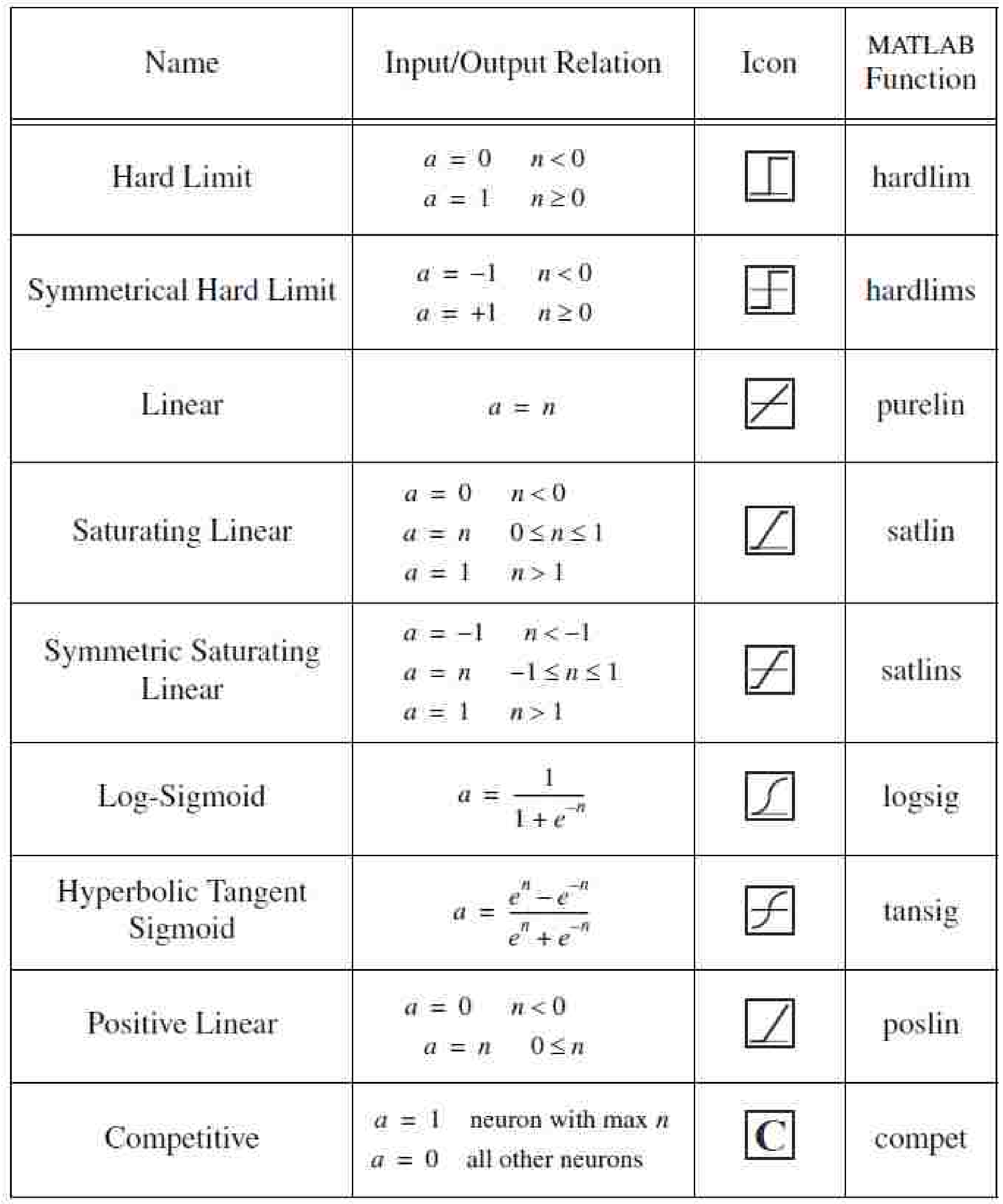
[70] G. Malkomes, C. Schaff, and R. Garnett, “Bayesian Optimization for Automated Model Selection,” *Adv. Neural Inf. Process. Syst. 29*, vol. 1, no. Nips, pp. 1–7, 2016.

[71] D. K. Duvenaud, “Automatic Model Construction with Gaussian Processes,” University of Cambridge, 2014.

[72] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, pp. 436–444, 2016.

[73] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, “Ensemble Selection from Librairies of Models,” in *Proceedings of ICML ’04*, 2004, vol. 34, pp. 1–21.

Appendix A



Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	C	compet

Figure 117: ANN transfer functions [38].