



Harnessing the Power of Artificial Intelligence (AI) for Transmission & Distribution Operations

Final Project Report

T-63

Power Systems Engineering Research Center
*Empowering Minds to Engineer
the Future Electric Energy System*



Harnessing the Power of Artificial Intelligence (AI) for Transmission & Distribution Operations

Final Project Report

Project Team

Yang Weng, Project Leader

Anamitra Pal

Arizona State University

Lang Tong

Cornell University

Graduate Students

Hamad Alduaij

Carl Morgenstern

Malhar Padhee

John Patterson

Jingyi Yuan

Arizona State University

Kursat Mestav

Xinyi Wang

Cornell University

PSERC Publication 21-08

September 2021

For information about this project, contact:

Yang Weng
Arizona State University
School of Electrical, Computer and Energy Engineering
Tempe, AZ, 85287
Phone: 650-924-3618
Fax: 480-965-3837
Email: yang.weng@asu.edu

Power Systems Engineering Research Center

The Power Systems Engineering Research Center (PSERC) is a multi-university Center conducting research on challenges facing the electric power industry and educating the next generation of power engineers. More information about PSERC can be found at the Center's website: <http://www.pserc.org>.

For additional information, contact:

Power Systems Engineering Research Center
Arizona State University
527 Engineering Research Center
Tempe, Arizona 85287-5706
Phone: 480-965-1643
Fax: 480-727-2052

Notice Concerning Copyright Material

PSERC members are given permission to copy without fee all or part of this publication for internal use if appropriate attribution is given to this document as the source material. This report is available for downloading from the PSERC website.

© 2021 Arizona State University. All rights reserved.

Acknowledgements

This is the final report for the Power Systems Engineering Research Center (PSERC) research project titled “Harnessing the Power of Artificial Intelligence (AI) for Transmission & Distribution Operations” (Project T-63).

The authors wish to thank the industry collaborators: Evangelos Farantatos (EPRI), Mahendra Patel (EPRI), Yingchen Zhang (NREL), Veda Santosh (NREL), Mark Westendorf (MISO), Matthew E Rhodes (SRP), Di Shi (GEIRINA), Zhe Yu (GEIRINA), Jianzhong Tong (PJM), Harvey Scribner (SPP), Baj Agrawal (APS), George Stefopoulos (NYPA), Atena Darvishi (NYPA), Honggang Wang (GE), Ming Jin (GE), Philip Hart (GE), Zhongyu Wu (MISO), Chen Wang (Dominion)

Executive Summary

Due to the increasing complexity of operations in transmission and distribution (T/D) grids, a number of scenarios have emerged where traditional decision-making tools have failed to control grid operations in a reliable manner. In this project, we explore if *physically meaningful artificial intelligence methods* can address this problem. We focus on three specific topics:

In Task 1, we perform prediction of equipment failure using several artificial intelligence (AI) techniques, including machine learning (ML) and matrix-based techniques, to assess localized power system asset health and provide prompt detection of asset degradation indicators. Through Task 2, we demonstrate seamless integration of physical laws using AI, which significantly improves the state-of-the-art state estimation and anomaly detection, when the system is unobservable due to the lack of sensor measurements. In Task 3, we move from monitoring to control for distribution grids, where lack of system information, due to unobservability and presence of unknown third-party controllers, is the primary concern. We show that AI can provide generalized power flow equations for robust monitoring and control of distribution grids, unavailable in the past when there are unobservability issue and third-party controllers.

Potential Benefits: Artificial intelligence (AI) and machine learning (ML) often offer solutions to problems that are traditionally considered intractable. In particular, this project demonstrates how novel ML techniques can use diverse sources of data streams to produce probabilistic forecasts of congestion and contingency patterns. The online learning offers several orders of magnitude reduction of computation costs. A framework for successfully integrating physical laws with failure patterns is also developed. Finally, the proposal shows how to draw inferences intelligently and efficiently from limited system information. The successful completion of the project result in the following specific benefits:

Benefits to RTOs/ISOs: Our AI-based learning algorithms helps RTOs/ISOs manage a diverse mix of resources reliably by analyzing a variety of scenarios that were not addressable in the past. The results of this research aids RTOs/ISOs in improving the performance of their state estimators and facilitating the integration of active renewable generation into RTOs/ISOs' system by enhancing situational awareness.

Benefits to utilities: Task 1 not only ensures optimal returns on investments made (e.g., purchasing expensive equipment such as a power transformer), but also keep the utility secure against unforeseen equipment failures, such as [1], by generating alarms in advance. Utilities can use the analysis carried out in Task 2 to model a grid that has limited observability. Additionally, utilities can maximize their control capability in a grid where the third-party controller is deployed without informing the utility in Task 3.

Benefits to vendors: Task 1 is extremely beneficial to vendors, because by generating alarms in advance, it creates an opportunity to run tests on the affected equipment (in coordination with the utility, e.g., during planned shutdowns), and repair/replace equipment parts, at a much lower cost. This also improves relations between vendors and utilities. Task 2 results in a distribution grid state estimator that is robust to missing system information. Task 3 provides controllability with

system unobservability. The three tasks are useful additions to modern distribution management systems. As a PSERC partner, GE shows great interest in the task's control design.

Summary of benefits:

- A method to forecast transformer failure based on phasor measurement unit (PMU) measurements
- A tool to extract system model for state estimation when systems are unobservable
- A control strategy to provide efficient distribution grid management with partial system information

Outcomes: This work leads to the development of data analytic tools that can be used by industry as well as academia. Specifically, the proposal's outcomes will be used for (a) *online monitoring of the health for power transformers*, (b) *distribution grid state estimation and control*. The investigators provided regular report to PSERC.

Industry Engagement: We had calls with our industry advisers to guide the work and discuss critical questions. We also seek out student internship opportunities with industry for more direct collaboration.

Potential Applications: One of the direct outcomes of this project is the creation of a tool that can predict the proximity to a failure of high voltage power transformers in advance. *Electric Power Group (EPG)*, an LLC that specializes in developing and implementing synchrophasor-based solutions and works closely with many PSERC industry members is especially interested in commercializing this tool.

Project Publications:

- [1] M. Padhee, R.S. Biswas, A. Pal, K. Basu, and A. Sen, "Identifying unique power system signatures for determining vulnerability of critical power system assets," ACM SIGMETRICS Perform. Eval. Rev., vol. 47, no. 4, pp. 8-11, Apr. 2020.
- [2] K. Mestav, J. Luengo-Rozas, and L. Tong, "Bayesian state estimation for unobservable distribution systems via deep learning," IEEE Transactions on Power Systems, vol. 34, no. 6, pp. 4910-4920, Nov. 2019.
- [3] K. Mestav and L. Tong, "Learning the Unobservable: High-Resolution State Estimation via Deep Learning," 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton).
- [4] K. Mestav and L. Tong, "State Estimation in Smart Distribution Systems with Deep Generative Adversary Networks," 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm).
- [5] K. Mestav and L. Tong, "Universal Data Anomaly Detection via Inverse Generative Adversary Network," IEEE Signal Processing Letters, vol. 27, pp. 511-515, March 2020.
- [6] X. Wang and L. Tong, "Innovations Autoencoder and its Application in One-class Anomalous Sequence Detection," submitted to Journal of Machine Learning Research, July 2021. Accessible online on arXiv at <https://arxiv.org/abs/2106.12382>

- [7] K. Mestav, X. Wang, and L. Tong, “A deep learning approach to anomaly sequence detection for high-resolution monitoring of power systems,” submitted for publication. Accessible online on arXiv at <https://arxiv.org/abs/2012.05163>
- [8] J. Yuan and Y. Weng, “Support Matrix Regression for Learning Power Flow in Distribution Grid with Unobservability,” IEEE Transactions on Power Systems, Aug. 2021.
- [9] J. Yuan, Y. Weng, and C. Tan, “Quantifying Hosting Capacity for Rooftop PV System in LV Distribution Grids,” International Journal of Electrical Power and Energy Systems, Jul 2021.

Student Theses:

- [1] K. Mestav, *A deep learning approach to state estimation and bad data detection*, Ph.D. Dissertation, January 2021.

Table of Contents

1. Introduction.....	1
1.1 Background	1
1.2 Report Organization	2
2. Overview of the Problem	3
2.1 Predicting Equipment Failure Using AI.....	3
2.2 Online Health Monitoring of Diverse Power System Equipment in Presence of Biased Data	11
2.3 Creation of a Generic Framework for Simultaneous Health Monitoring of Diverse Power System Assets	19
2.4 Completion of a Generic Framework for Simultaneous Multiple-Substation Health Monitoring of Diverse Power System Assets.....	22
2.5 Future Work: Automation of Event Detection in Simultaneous Multiple-Substation Health Monitoring of Diverse Power System Assets	26
3. Deep Learning Approach to Power System State Estimation and Bad-Data Detection.....	28
3.1 Introduction.....	28
3.1.1 Bayesian State Estimation of Unobservable Distribution System via Deep Learning.....	28
3.1.1.1 Related Work.....	29
3.1.1.2 Summary of Results and Contributions.....	30
3.1.2 A Deep Learning Approach to Anomaly Sequence Detection.....	31
3.1.2.1 Related Work.....	31
3.1.2.2 Summary of Approach and Contributions.....	33
3.2 Bayesian State Estimation via Deep Learning.....	34
3.2.1 Bayesian State Estimation and Neural Network	34
3.2.1.1 Algebraic and Statistical Models.....	34
3.2.1.2 Bayesian Stat Estimation.....	34
3.2.1.3 Neural Network Approximation.....	35
3.2.2 Learning with Deep Neural Network	36
3.2.2.1 Training the Neural Network.....	36
3.2.2.2 Learning Net Injection Distributions.....	37
3.2.3 Bad Data Detection and Data Cleansing	38
3.2.4 Simulations Results and Discussions	39
3.2.4.1 Simulation Settings.....	39

3.2.4.2	Distribution Learning	39
3.2.4.3	Simulation Results.....	40
3.2.4.4	Optimizing the Structure of Feep Neural Network.....	43
3.2.5	Conclusion.....	44
3.2.6	Appendix	45
3.3	Anomaly Data Sequence Detection via ICA-GAN	46
3.3.1	System and Anomaly Detection Model.....	46
3.3.1.1	Measurement and Anomaly Models.....	46
3.3.1.2	Sensor Level Anomaly Sequence Detection	47
3.3.1.3	Distributed Detection and Data Fusion Model	48
3.3.2	Sensor-Level Anomaly Sequence Detection	48
3.3.2.1	Anomaly Sequence Detection via Uniformity Test	49
3.3.2.2	Extracting Independent Components via ICA-GAN.....	50
3.3.3	System-level Anomaly Detection.....	52
3.3.4	Numerical Case Studies.....	53
3.3.4.1	Benchmark Techniques and Implementations.....	54
3.3.4.2	Case I: System Anomalies in CPOW Measurements.....	55
3.3.4.3	Case II: Natural Anomalies in PMU Measurements	56
3.3.4.4	Case III: Unobservable Attacks on PMU State Estimation.....	58
3.3.5	Discussions	60
3.3.5.1	Anomalies vs. System Dynamics	60
3.3.5.2	Anomalies in Training Data	60
3.3.5.3	Offline vs. Online Training	60
3.3.6	Conclusion.....	61
4.	Online Learning Distribution Voltage Controller with Convergence.....	62
4.1	Introduction to graph neural networks.....	62
4.2	Problem Setting.....	64
4.3	Linear Voltage Controller	65
4.4	Asymptotic Convergence Guarantees.....	67
4.4.1	Policy for Asymptotic Convergence of Linear Self-Tuning Regulators	68
4.4.2	Conditions for Strongly Consistent Least Square Estimates	68
4.4.3	Ensuring Asymptotic Convergence.....	70
4.5	Empirical Validation.....	72

4.5.1	Testing Asymptotic Convergence	73
4.5.2	Evaluating Converged Regulation.....	74
4.5.3	Comparing Scaling Properties of the Regulators	75
4.5.3.1	Examining How Observability Affects System Regulation	75
4.5.3.2	How Controllability Affects Regulation	76
4.6	Conclusion and Future Work	78
5.	Conclusion and Future Work	79
Appendix A: SNR_STD_exporter.py		80
Appendix B: SZS_validate.py		84
Appendix C: SZS_classify.py		89
References		93

List of Figures

Figure 2.1 An operating distribution transformer showing visual indications of possible degradation.....	4
Figure 2.2 Line current 3 days before a lightning arrester fault.	5
Figure 2.3 Line current 12 hours before and after a lightning arrester fault.....	5
Figure 2.4 Process for obtaining SNR bandwidth limit counts from PMU data.	6
Figure 2.5 Background limit counts for 12 PMU measurement inputs.	8
Figure 2.6 Limit counts approaching the time of the failure event.....	9
Figure 2.7 Limit counts just before the time of the failure event.	9
Figure 2.8 Change in SNR Standard Deviation Limit Count for the Arrester Failure, 3-6 Hours Pre-Fault to 0-3 Hours Pre-Fault.....	10
Figure 2.9 Z-score concept [3].....	12
Figure 2.10 Moving-window sum of Z-scores technique.	13
Figure 2.11 Real power system event data (lightning arrester) processed through sum of Z-scores.....	14
Figure 2.12 More subtle trends extracted from bushing failure data by the sum of Z-scores method.....	15
Figure 2.13 Flowchart of the SVM algorithm training and evaluation methodology.	16
Figure 2.14 Timestamp-based labeling of input training and testing data.....	18
Figure 2.15 Accuracy of SVM model on 30% randomly selected testing data (no training data used for testing) for lightning arrester event.	18
Figure 2.16 Accuracy of SVM model on 30% randomly selected testing data (no training data used for testing) for transformer bushing event.	19
Figure 2.17 Features and outputs of "SNR_STD_exporter.py".....	20
Figure 2.18 Features and outputs of "SZS_validate.py".....	20
Figure 2.19 Features and outputs of "SZS_classify.py".	21
Figure 2.20 Changes in the SNR STD profile after a fault due to changing system operating point.	22
Figure 2.21 Testing of Matrix Profile for detecting anomalies in periodically varying SNR STD data on 48 hours' data for W=3000, W=10000, and W=30000. Vertical axis represents distance between motifs and horizontal axis is time-series sample number	23
Figure 2.22 Flow diagram of derivative difference method	24
Figure 2.23 Derivative difference method in action on real power system data	24
Figure 2.24 Multiple derivative differences combined into a series of 2D "heatmap" frames.....	25
Figure 2.25 Evolution of heatmap frames prior to catastrophic asset failure	25

Figure 2.26 Concept of ML-based operational decision-making based on derivative difference heatmap data	26
Figure 2.27 On the left is a simple GAN model and on the right are time-series generated samples [8]	27
Figure 3.1 Multi-layer Forward Neural Network.	35
Figure 3.2 Estimated cumulative distribution and 95% confidence interval.	40
Figure 3.3 ASE performance for three state estimators. Left: ASE (in absence of bad data) of estimators in different hours. Right: ASE at hour 17 in the presence of bad data of different strength; Line 1 shows ASE without bad data, Line 2 shows ASE with bad data and Line 3 shows ASE with bad data and cleansing for estimators.	41
Figure 3.4 NN Architecture - Total number of neurons fixed.	43
Figure 3.5 A schematic of a distributed anomaly detection system.	48
Figure 3.6 A schematics of ICA-GAN for anomaly sequence detection.....	49
Figure 3.7 Learning structure of ICA-GAN.	51
Figure 3.8 Learning structure of ICA-GAN.	55
Figure 3.9 ROC curves (TPR at FRP=0.05). Top left: Detector at A (TPR=0.2564). Top right: Detector at B (TPR=0.3077). Bottom left: Detector at the bus (TPR=0.1053). Bottom right: Detector at the control center (TPR=0.7368.).....	56
Figure 3.10 ROC curves for anomaly case 1. Left: The EPFL System. Right: The Synthetic North Texas system.....	57
Figure 3.11 ROC curves for anomaly case 2. Left: The EPFL System. Right: The Synthetic North Texas system.....	59
Figure 4.1 The change in voltage data from a year of unregulated hourly operation at a measured bus. Fitted with a Gaussian based on the expected value and variance of the data, $\mathbb{E}(\Delta V) = 0.036$ and $\sigma^2 = 3636.62$	69
Figure 4.2 The feeders used for our experiments.	72
Figure 4.3 The overarching figure shows the average regret of the linear and quadratic models <i>without</i> feedback delay. The smaller zoomed-in average regret shows that the linear model has a lower average after almost a year of hourly operation. On the contrary, the instantaneous regret snapshot shows that once converged, the quadratic model, regulates the system better.	73
Figure 4.4 Similar to Figure 4.3, the overarching figure shows the average regret of the linear and quadratic models.	74
Figure 4.5 The R^2 scores for the DLC (a) and IEEE 123-bus (b) feeder based on the system loading when estimating the voltage.	75
Figure 4.6 Figures (a) and (b) show the final average voltage deviation from 250 stages on the IEEE 123-bus feeder, according to the number of observable busses. Figure (c) shows the number stages of initial random exploration require to establish a stable estimate.....	76

Figure 4.7 Minimum values of τ^* , where <i>lin</i> refers to the linear model (4.6), <i>quad</i> refers to the homogeneous quadratic model, and <i>previous</i> refers to 3 days worth of initial exploration roughly recommended by the previous methods [85], [86].....	77
Figure 4.8 The normalized average regret, $(R(n)/n)/k$, for the DLC feeder with hourly feedback delay.	77
Figure 4.9 The normalized average regret, $(R(n)/n)/k$, for the DLC feeder without feedback delay.	77

List of Tables

Table 2.1 Pre-fault current SNR standard deviation samples exceeding 2 dB.	7
Table 2.2 Pre-fault voltage SNR standard deviation samples exceeding 2 dB.....	7
Table 3.1 Bad data detection and cleansing with varying strength of bad data. $\eta = 0.3$	42
Table 3.2 Pruning simulation results with the threshold = 0.0005	44
Table 3.3 TPR values of different algorithms at FPR=0.05 constraint under bad data and data attack anomalies.	58
Table 3.4 Average squared error of state estimation.	60

1. Introduction

1.1 Background

Due to the increasing complexity of operations in transmission and distribution (T/D) grids, a number of scenarios have emerged where traditional decision-making tools have failed to control grid operations in a reliable manner. Therefore, is it possible to address this problem by exploring artificial intelligence methods that are physically meaningful.

First, the equipment and infrastructure comprising the modern electric power system must constantly operate at a satisfactory level of performance to ensure that the system is able to safely deliver electricity to customers. The components of the electric power system, commonly referred to in industry nomenclature as *power system assets*, consist of generators, transformers, arresters, switchgear, conductors, insulators, monitoring devices, and numerous other key infrastructural components placed throughout the system, all of which must be maintained in order to ensure reliable system operation. As power system assets age, they are prone to degradation which can not only result in decreased performance, but also lead to sudden, catastrophic, and potentially deadly failure events. For example, a large power transformer in Avondale, Arizona exploded violently in 2016, resulting in a large fire and considerable release of smoke into the environment [1]. Other widely-reported asset failures include the explosive failure of a potential transformer and multiple insulator failures, some of which have been implicated in the ignition of serious wildfires [2]. Therefore, can we conduct equipment diagnosis by forecasting its working conditions?

Second, situational awareness is quite important especially in power distribution systems with limited measurements. Specifically, we are interested in power system state estimation and anomaly detection when the system is unobservable due to the lack of sensor measurements. Such motivation comes from greater presence of distributed energy resources, especially in the form of inverter and converter-based resources that make the power grid increasingly stochastic, dynamic, and vulnerable to system/cyber anomalies. Therefore, can we conduct (i) state estimation of an unobservable distribution system and (ii) detection of anomaly sequence in high-resolution measurements?

Third, the distribution system is widely unobservable and unmodeled, so model-based optimization techniques are not presently feasible for voltage regulation. Past model-free strategies are either too simple (rule-based), cannot operate the present devices for voltage regulation (capacitors), and/or have unrealistic assumptions about the communication speed and data resolution. To overcome these issues, can we introduce online learning controllers that adjusts the devices on a distribution feeder as the loading changes, while avoiding perturbing the system too many times?

To solve these three questions above, we focus on three specific topics: In Section 2, we perform prediction of equipment failure using several artificial intelligence (AI) techniques, including machine learning (ML) and matrix-based techniques, to assess localized power system asset health and provide prompt detection of asset degradation indicators. Through Section 3, we demonstrate seamless integration of physical laws using AI, which significantly improves the state-of-the-art

state estimation and anomaly detection, when the system is unobservable due to the lack of sensor measurements. In Section 3, we move from monitoring to control for distribution grids, where lack of system information, due to unobservability and presence of unknown third-party controllers, is the primary concern. We show that AI can provide generalized power flow equations for robust monitoring and control of distribution grids, unavailable in the past when there are unobservability issue and third-party controllers.

1.2 Report Organization

This report is organized into five chapters. The first chapter discussed the background of the problem and objectives for new machine learning-based methods. The second chapter conducts prediction of equipment failure using several artificial intelligence (AI) techniques. The third chapter provides solutions on state estimation and anomaly detection for systems with unobservability. Moving from monitoring to control, the fourth chapter shows an online learning method for controls with performance guarantees. Finally, the fifth section concludes the project and discusses future scope of the project.

2. Overview of the Problem

2.1 Predicting Equipment Failure Using AI

In order to extract the maximum economic value from their assets, it is desirable for utilities to operate assets for as long as possible before replacing them. As depicted in Figure 2.1, many power system assets in the field remain operable even with considerable age or visual signs of deterioration. As such, repair or replacement of assets as informed by *electrical measurements*, rather than age or visual inspection, may provide considerable advantage to electric power system stakeholders.

A previous investigation (PSERC project S-74) indicated that by calculating the moving-window standard deviation of the signal-to-noise ratio (SNR) of power system current and voltage data, it may be possible to extract indications of degradation for certain types of assets. In this project, the previous work in this field was extended to encompass several new strategies for asset health diagnosis, including the use of artificial intelligence (AI) to identify and amplify trends in the data that are potentially indicative of asset degradation. Newly-acquired datasets from multiple power system asset failures were processed and analyzed for these techniques, and multiple options for AI-based asset failure prediction were developed, and their performance was evaluated using real-world data.

By applying the previously developed standard deviation of SNR methodology from PSERC project S-74 to new data from a lightning arrester failure, the plot of Figure 2.2 was produced. Figure 2.2 was generated using a dataset containing time-series phasor measurement unit (PMU) data including 3 days' worth of data prior to the lightning arrester failure event. The red, blue, and green plots represent data from the A-, B-, and C-phase line current values measured over time, respectively. The top row of plots in Figure 2.2 is the original data, the second row is the original data with any data dropouts (due to PMU errors or unreported values) removed, the third row is the time-series SNR of the data, and the bottom row is the time-series standard deviation of the SNR data.



Figure 2.1 An operating distribution transformer showing visual indications of possible degradation.

One notable feature of the SNR standard deviation data for this particular event is the prevalence of transient increases in SNR standard deviation circled in red in Figure 2.2. Although these transient increases are not a definitive indication of asset degradation on their own, they do suggest that the variability in line current noise becomes markedly higher as the failure event is approached.

By restricting the time period of data analyzed to 12 hours before and after the time of the lightning arrester fault, the plots depicted in Figure 2.3 were obtained. These plots further emphasize the abnormality of these pre-event transient increases in the SNR standard deviation. Although transient increases in the SNR standard deviation do occur throughout the measured data set, their frequency was qualitatively higher in close temporal proximity to the failure event. In light of the qualitative observation about the evolving frequency of transient SNR standard deviation increases, it was deemed prudent to develop a quantitative methodology to determine the relative frequency of these events.

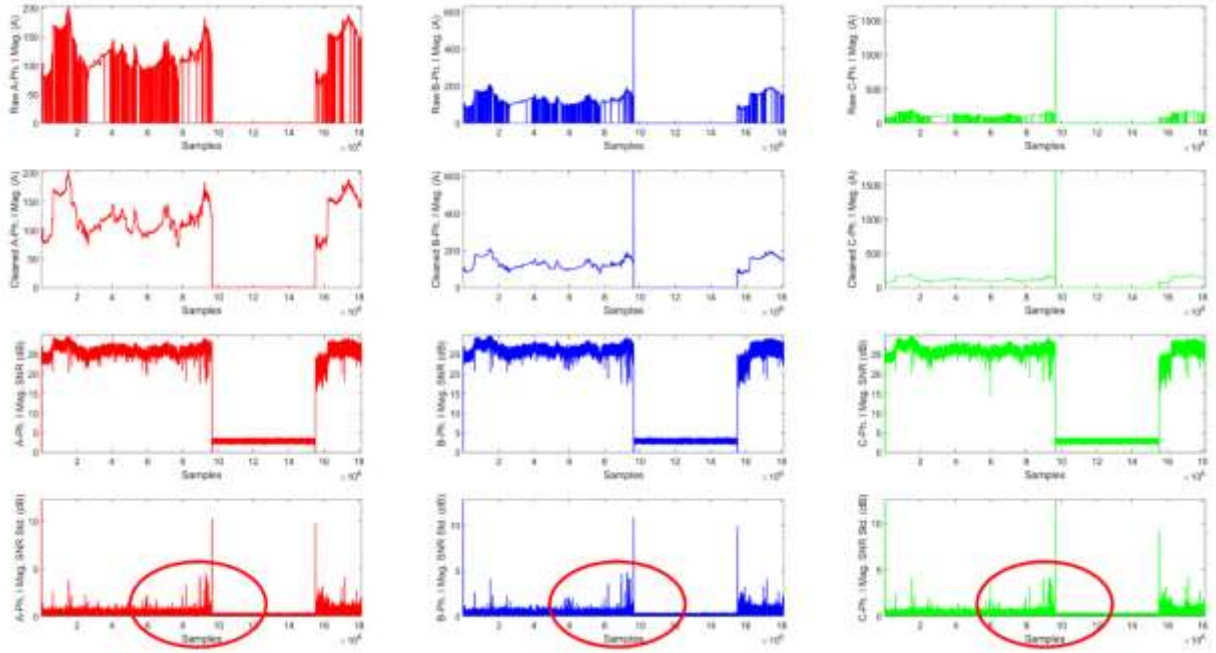


Figure 2.2 Line current 3 days before a lightning arrester fault.

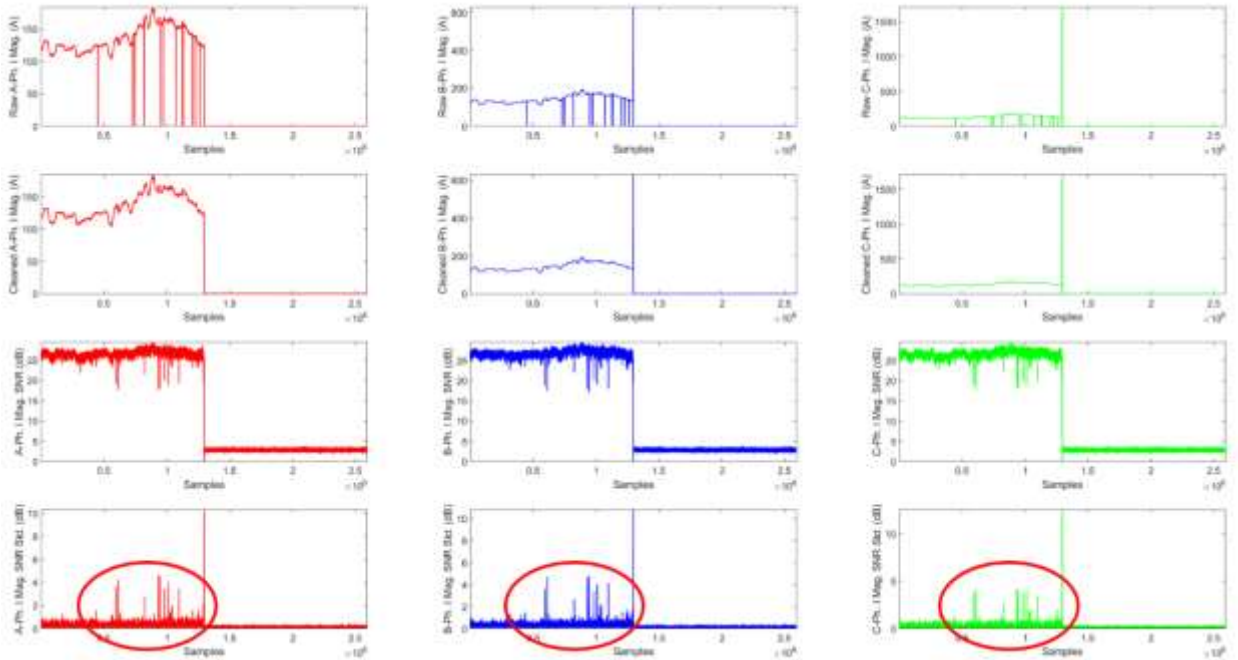


Figure 2.3 Line current 12 hours before and after a lightning arrester fault.

In order to quantitatively estimate the frequency of transient SNR standard deviation increases in any set of PMU data, the algorithm summarized by the flow diagram of Figure 2.4 was developed.

For each PMU data file provided for the event, the data of interest was preprocessed, the SNR and SND standard deviation values were obtained, and the data was initially plotted in a format similar to that shown in Figure 2.2 and Figure 2.3.

An additional step was performed to automatically count the number of PMU data samples whose SNR standard deviation exceeded a set threshold, which was defined by visual inspection of the above plots to be 2 decibels (dB). If a large number of samples with high SNR standard deviation are detected just before the time of an asset failure event, and if relatively few samples with high SNR standard deviation were detected long before or shortly after the time of the event, then this is evidence to suggest that a relationship exists between the frequency of these transients and the particular event occurring.

The algorithm of Figure 2.4 can be applied to one set of data at a time or can be automated to operate on numerous data sets simultaneously. If multiple data sets are handled simultaneously, then a set of bar plots showing limit count values in a given time period can also be created, thus leveraging the capabilities of modern computer hardware to provide novel insight into the behavior of electric power systems over time.

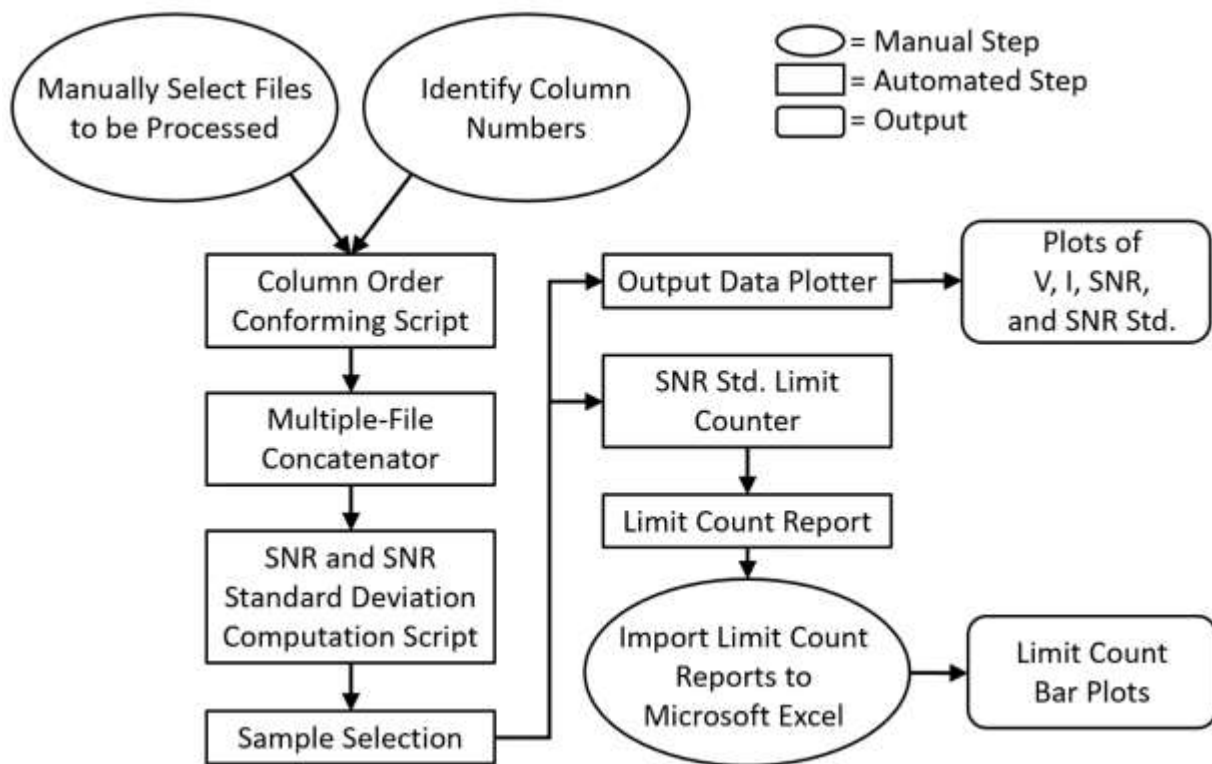


Figure 2.4 Process for obtaining SNR bandwidth limit counts from PMU data.

The process described in Figure 2.4 was initially applied to the measurements of the line current and bus voltage from one PMU in close proximity to the site of the lightning arrester failure. A count of the number of PMU-scale data points with SNR standard deviation values exceeding 2 dB (a manually-selected threshold) was taken for several 6-hour periods of data prior to the event of interest. The number of "limit-exceeding" data points for each current phase in every period of

time was quantified in Table 2.1, and the number of "limit-exceeding" data points for each voltage phase in each period of time was quantified in Table 2.2.

Table 2.1 Pre-fault current SNR standard deviation samples exceeding 2 dB.

	A-phase	B-phase	C-phase
0-6 hours pre-fault	2660	2759	2559
6-12 hours pre-fault	627	635	626
12-18 hours pre-fault	342	273	264
18-24 hours pre-fault	0	0	0
24-30 hours pre-fault	0	0	0
30-36 hours pre-fault	124	184	312
36-48 hours pre-fault	0	0	0

Table 2.2 Pre-fault voltage SNR standard deviation samples exceeding 2 dB.

	A-phase	B-phase	C-phase
0-6 hours pre-fault	1909	1951	1690
6-12 hours pre-fault	545	520	550
12-18 hours pre-fault	61	46	46
18-24 hours pre-fault	152	140	151
24-30 hours pre-fault	95	132	128
30-36 hours pre-fault	636	613	833
36-48 hours pre-fault	0	100	0

From Table 2.1, it is clear that the frequency of high-standard deviation samples derived from the current measurement trends upward as the time of the lightning arrester fault approaches. Although this trend is less prominent in the "limit counts" computed for the voltage measurements, a striking increase in the number of limit-exceeding SNR standard deviation values is also present in the final 6 hours of voltage data prior to the time of the fault event.

If this methodology were to be applied to a real-world asset health prediction application, it would be desirable to not only know *if* a power system asset were degrading, but also *where* it is degrading. In order to localize the source of these transient SNR standard deviation samples, the same process may be applied to a large number of PMU data sources. Figure 2.5 – Figure 2.7 depict both voltage and current "limit counts" from 12 PMUs in the geographical vicinity of the lightning arrester failure analyzed in Figure 2.2 and Figure 2.3. To better reflect the observed behavior of this power system, the limit threshold for voltage SNR standard deviation was set to 4.

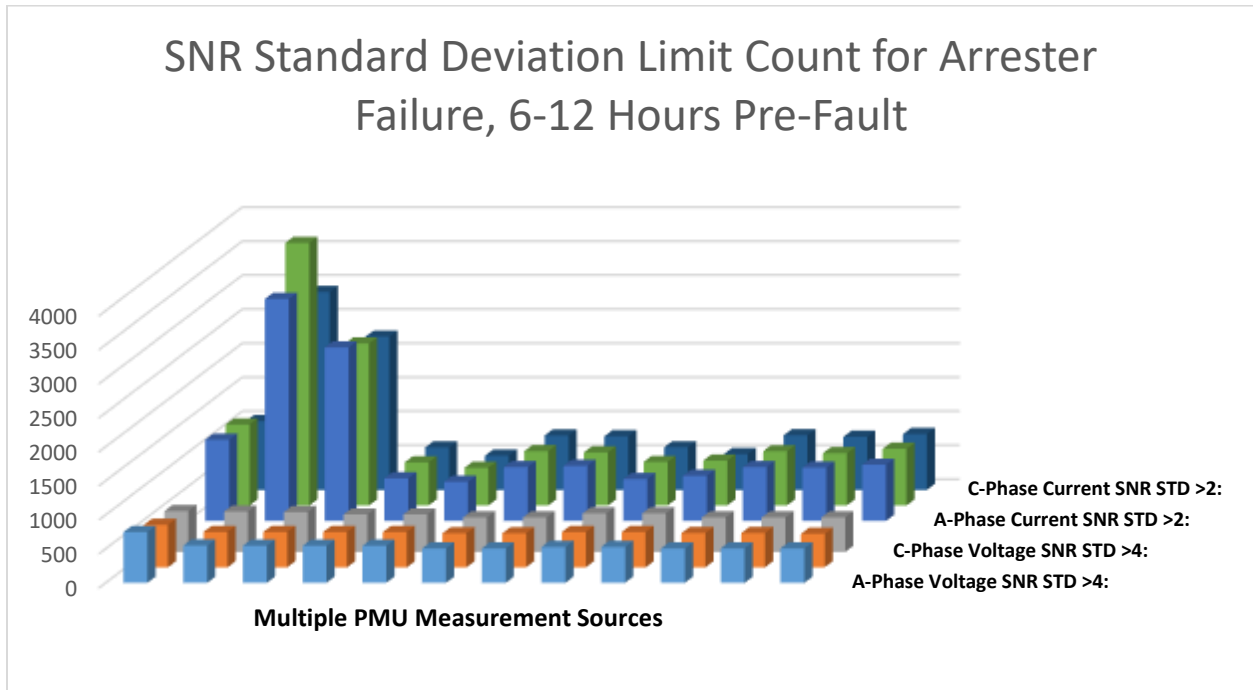


Figure 2.5 Background limit counts for 12 PMU measurement inputs.

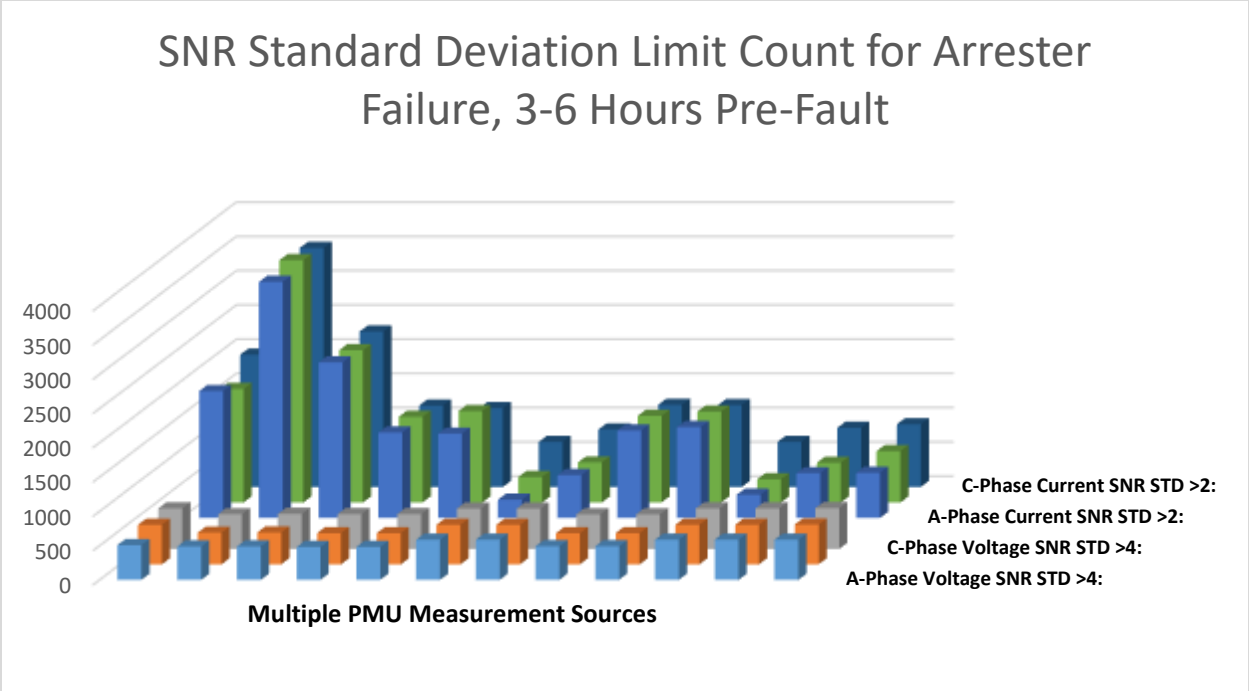


Figure 2.6 Limit counts approaching the time of the failure event.

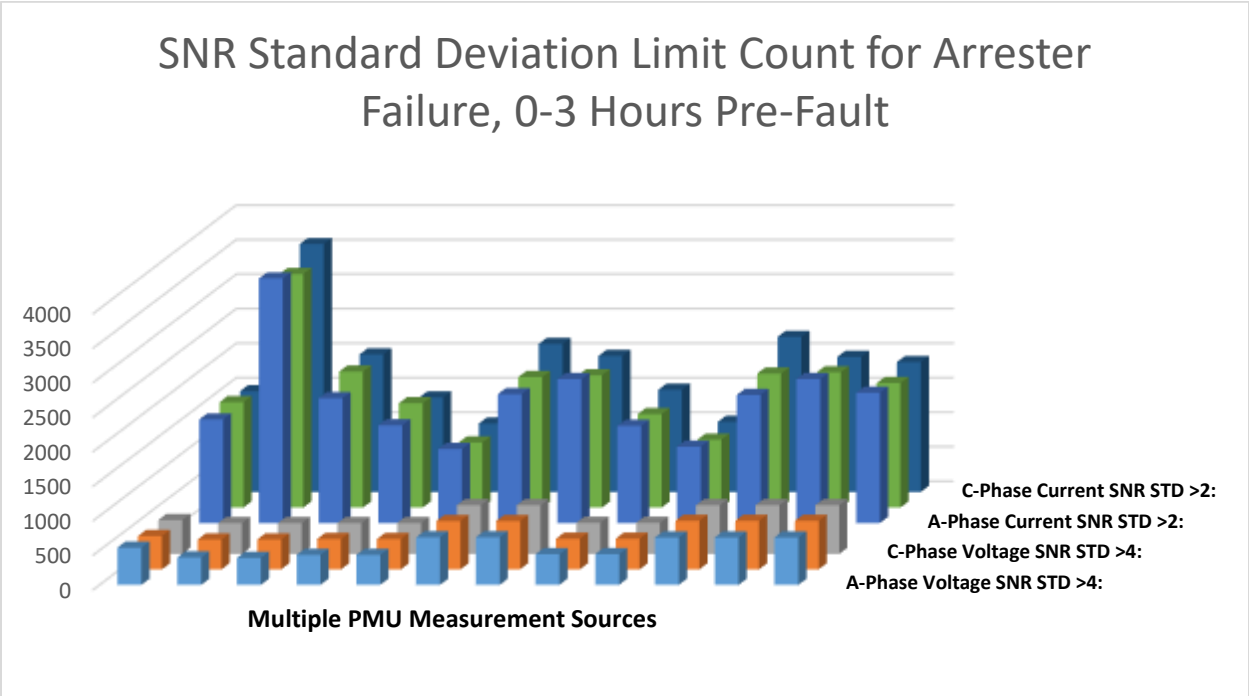


Figure 2.7 Limit counts just before the time of the failure event.

Figure 2.5 quantifies the number of SNR standard deviation samples exceeding the defined threshold for each of 12 PMUs of interest during a period of 6 to 12 hours before the incidence of the lightning arrester failure. This time range provides a "baseline" value for the typical frequency

with which limit-exceeding values are detected well before the fault event. Figure 2.6 indicates the frequency of limit-exceeding values in the period of 3 to 6 hours before the event, and Figure 2.7 indicates the frequency of limit-exceeding values in the period of 0 to 3 hours before the event.

It is clear from all three plots that certain PMUs produce more frequent "limit-exceeding" measurements than others. It is additionally evident that certain PMUs show a significantly greater increase in limit count as the event is approached than others. As such, rather than analyzing the *absolute* number of limit-exceeding measurements as a means for detecting asset health degradation, it may instead be more useful to track the *evolution* of number of limit-exceeding measurements over time.

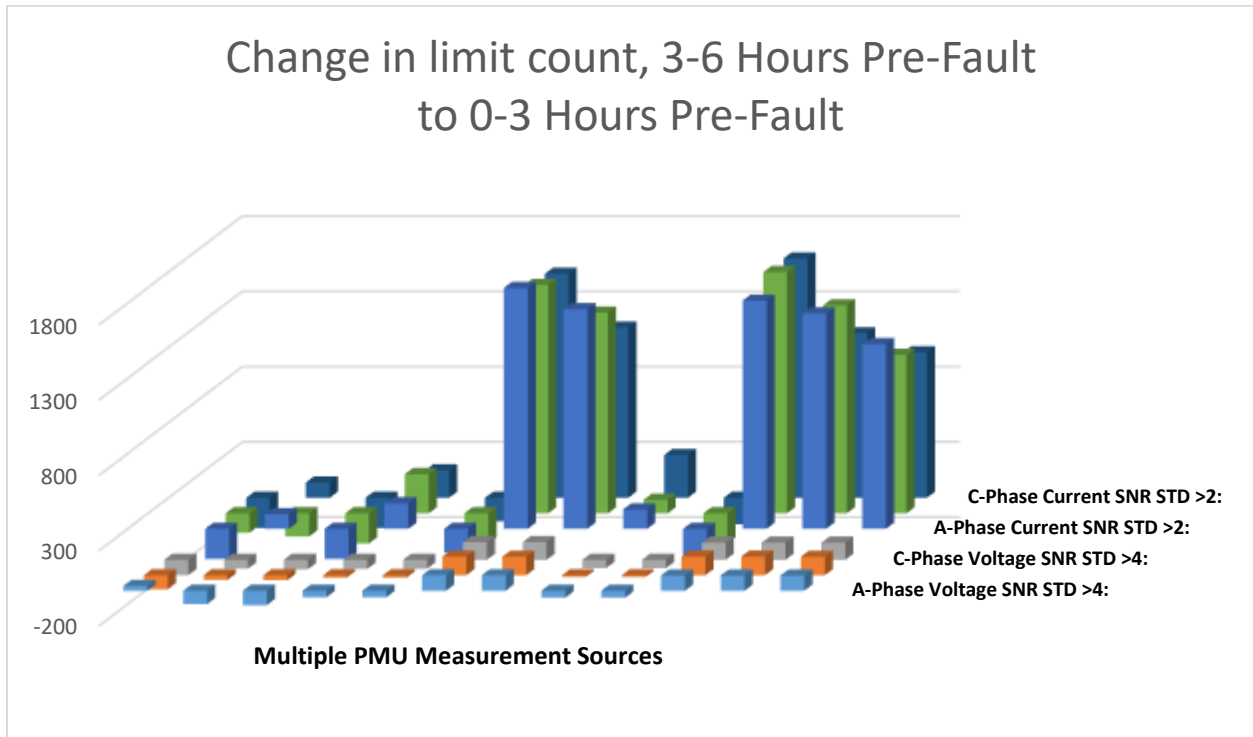


Figure 2.8 Change in SNR Standard Deviation Limit Count for the Arrester Failure, 3-6 Hours Pre-Fault to 0-3 Hours Pre-Fault.

Figure 2.8 depicts the *difference over time* in the number of limit-exceeding SNR standard deviation values computed from each PMU from *3-6 hours* pre-fault to *0-3 hours* pre-fault. This serves as an indication of the *local evolution* of the SNR standard deviation exhibited by this region of the power system. From the plot of Figure 2.8, it is clearly evident that the current measurements of 5 specific PMUs exhibit a much greater increase in the frequency of transient SNR standard deviation measurements than the other PMUs being analyzed. The power system assets in close proximity to these PMUs would be prime candidates for immediate inspection for degradation if this analysis were performed on real-time data and such results were observed.

2.2 Online Health Monitoring of Diverse Power System Equipment in Presence of Biased Data

Although the asset degradation algorithm and techniques described in Section 1.1 showed some promise in providing an indication of localized asset degradation over time, they suffered from several drawbacks that make them less-than-suitable for use in a real-time power system environment. An improved algorithm for analyzing PMU data and extracting relevant features that are indicative of degrading asset health would ideally have the following additional capabilities:

- It should provide an estimate of asset health that is immediate (within seconds), rather than an estimate that lags the inbound measurements by multiple hours
- It should automatically adjust criteria for "limit-exceeding" values, rather than relying on manual specification of these thresholds
- It should respond strongly to abnormal SNR standard deviation values and should largely ignore normal or baseline SNR standard deviation values

In response to these opportunities for improvement, a number of different methodologies for improving the algorithm were considered. The technique that was proposed, hereafter referred to as the *sum of Z-scores* method, or SZS method, was developed to address these challenges. A detailed description of the SZS method will follow after a brief review of the concept of the statistical Z-score and its meaning.

In statistics, the *Z-score* (or *standard score*) provides a measure of the relative closeness of a particular measurement sample to the mean of the data from which it is sampled [3]. In particular, the Z-score is the *number of standard deviations from the mean* at which a particular sample lies. The Z-score concept assumes a Gaussian distribution of measurement values. A conceptual overview of the Z-score and its relationship to the Gaussian "bell curve" is depicted in Figure 2.9.

As shown in Figure 2.9, the probability of randomly selecting a sample with a Z-score close to zero is much greater than the probability of randomly selecting a sample with a large positive or negative Z-score. Furthermore, if the Z-scores of a large number of samples are added together, they should approach zero as the number of samples measured increases. In other words, if a data set with Gaussian distribution, a fixed mean, and a fixed standard deviation is measured repeatedly, the *sum of Z-scores* measured should approach zero [4].

The tendency for the sum of Z-scores of a large number of measured samples to approach zero makes such a sum a good indicator of changes in the underlying distribution of a data set. If a long-term mean and long-term standard deviation of measured values are used to perform Z-score calculations on incoming data, and if the sum of these Z-scores over time deviates from zero, this indicates a change in the underlying distribution of the data. This phenomenon is leveraged to provide a long-term moving-window sum of Z-scores, thereby allowing the *sum of Z-scores* method to detect deviating SNR standard deviation values or other time-series values.

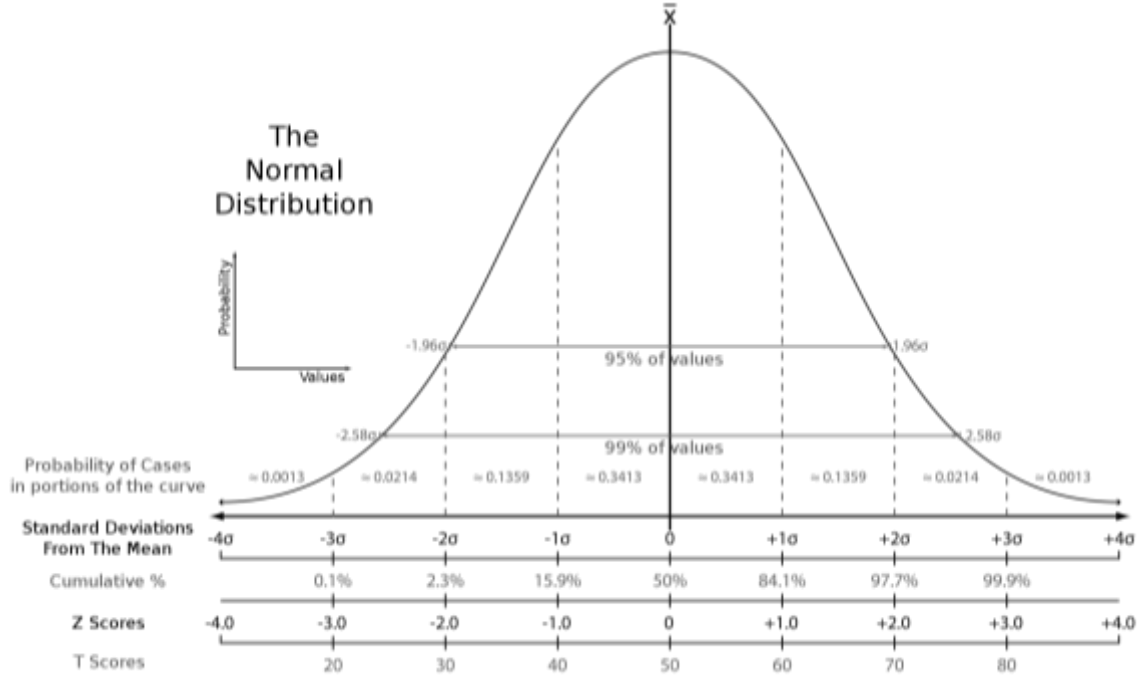


Figure 2.9 Z-score concept [3].

The mathematical formulation of the Z-score is expressed as **Error! Reference source not found.:**

$$z = \frac{x - \mu}{\sigma} \quad (1)$$

where, z is the z-score of a given sample x , μ is the mean (average) of the data set, and σ is the standard deviation of the data set.

The sum of Z-scores over an arbitrary window from the data point number m to the data point number n may therefore be expressed as:

$$Z = \sum_{i=m}^n \frac{x_i - \mu}{\sigma} \quad (2)$$

where x_i is the i -th data point in the range of data points m to n .

The value of the sum of Z-scores over a sliding window at a given data point n in the data set may therefore be expressed as in (3):

$$Z_m = \sum_{i=m}^n \frac{x_i - \mu}{\sigma} \quad (3)$$

Using a similar sliding window procedure, the value of the SNR (in decibels) over a sliding window of $(n - m + 1)$ values at a given data point n in the data set may be expressed as [5]:

$$SNR_m = 10 * \log \left(\frac{\mu}{S} \right) \quad (4)$$

where μ is the mean of the sliding window data and S is the standard deviation.

The *sum of Z-scores* methodology satisfies the suggested criteria listed at the beginning of this report section as follows:

- The values reported by the SZS method are sample-by-sample; the evolution of any power system measurement can be determined at any point in time
- The assumed mean and standard deviation of the data being analyzed is constantly being updated as new SZS values are computed; therefore, the sensitivity of the SZS method to anomalies is automatically adjusted as new data arrives
- The SZS method strongly amplifies abnormal (outlier) measurements, as they have disproportionately high Z-scores; measurements close to the mean of the data are essentially ignored by the SZS method

In summary, the SZS method provides an immediate, self-adjusting method for detecting anomalous measurements in power system data. A toy example of the SZS method being applied to some test data is depicted in Figure 2.10. Within Figure 2.10, the top plot shows raw random data with 4 intentionally-added outliers. The middle plot shows the calculated Z-core of each data point (note that the outliers are over 5 standard deviations away from the mean!). Finally, the third plot shows the moving-window sum of the last five Z-scores measured. From the third plot, it is evident that transient anomalies persistently increase the SZS, thus providing a method to test for recent anomalous behavior, even when such behavior is not occurring at that instant.

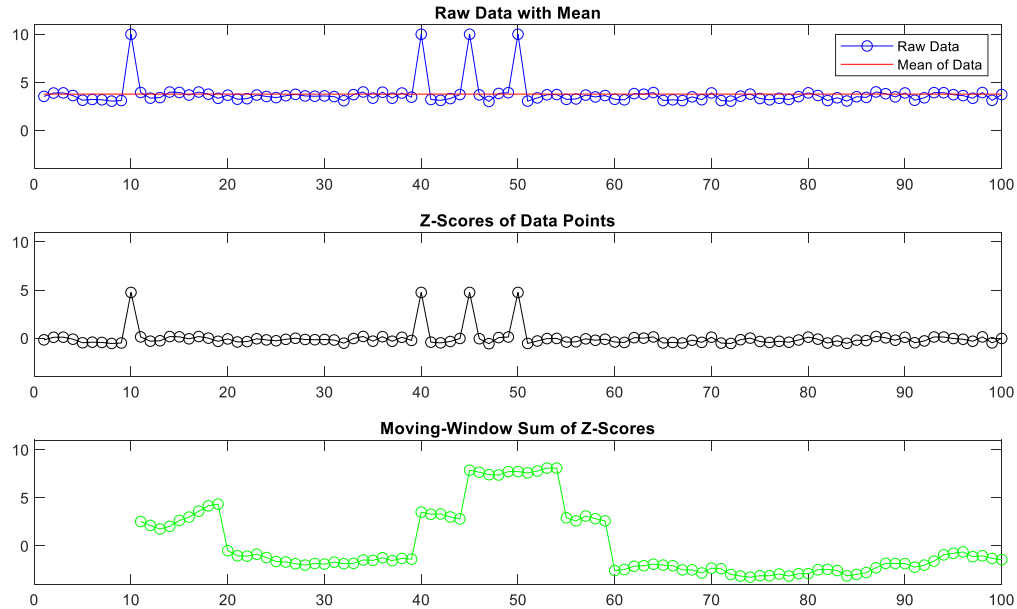


Figure 2.10 Moving-window sum of Z-scores technique.

When applying the described SZS method to PMU data acquired from a real-world power system event, the ability of the SZS method in tracking transient outlier phenomena over time becomes even more evident. Figure 2.11 depicts SNR standard deviation data captured approximately 40 hours before a lighting arrester failure, as well as the SZS values calculated for these data points. The SZS data is consistently positive, due to the many small transient events present throughout

the data. Several hours before the fault occurs, it becomes evident that a sharp increase in the SZS values obtained for the current measurements is present. Had this same observation been made in the hours leading up to the arrester failure, mitigating action might have been possible to prevent the failure event altogether.

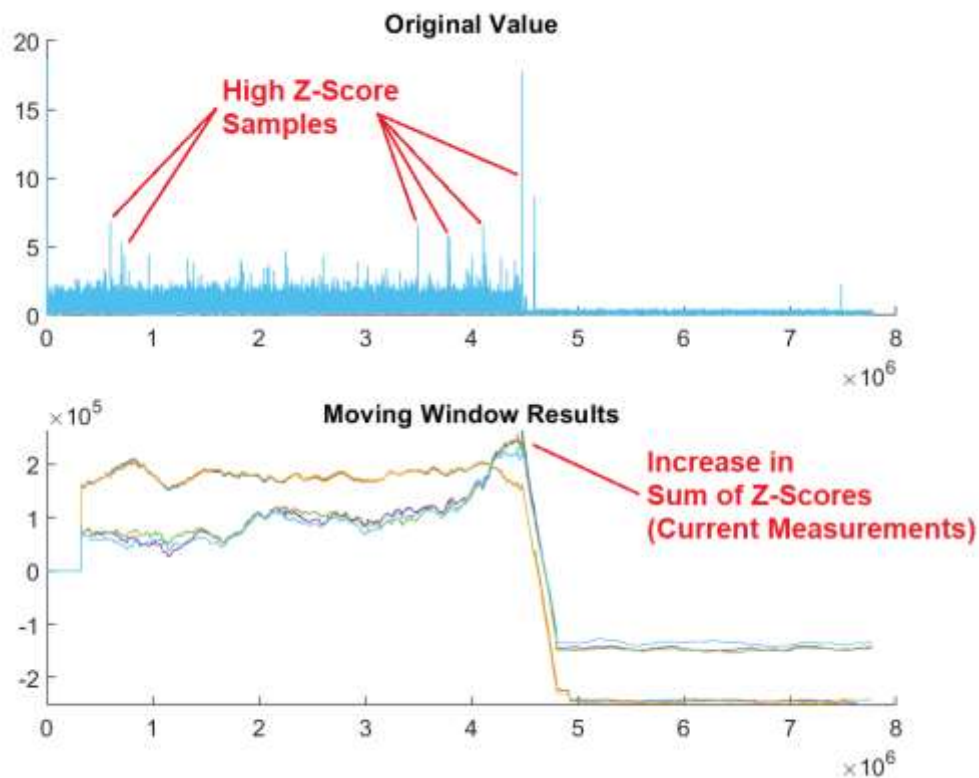


Figure 2.11 Real power system event data (lightning arrester) processed through sum of Z-scores.

Some power system events, such as the transformer bushing failure analyzed in Figure 2.12, showed much more subtle changes in the SZS prior to the event; however, detectable changes were still present. Within Figure 2.12, both positive and negative SZS values are present, suggesting a relatively stable system with few large spikes. Closer to the time of the event, however, the SZS values begin to sweep more aggressively positive, then negative over time. When the fault event finally does occur, a sharp increase in SZS associated with the current spike at the event time is seen. Shortly thereafter, as the moving window slides past the time of the fault, the SZS promptly falls to a low value, suggesting a sudden decrease in the height of the SNR standard deviation values post-fault that is consistent with previous correlation of these values with degrading assets in PSERC project S-74.

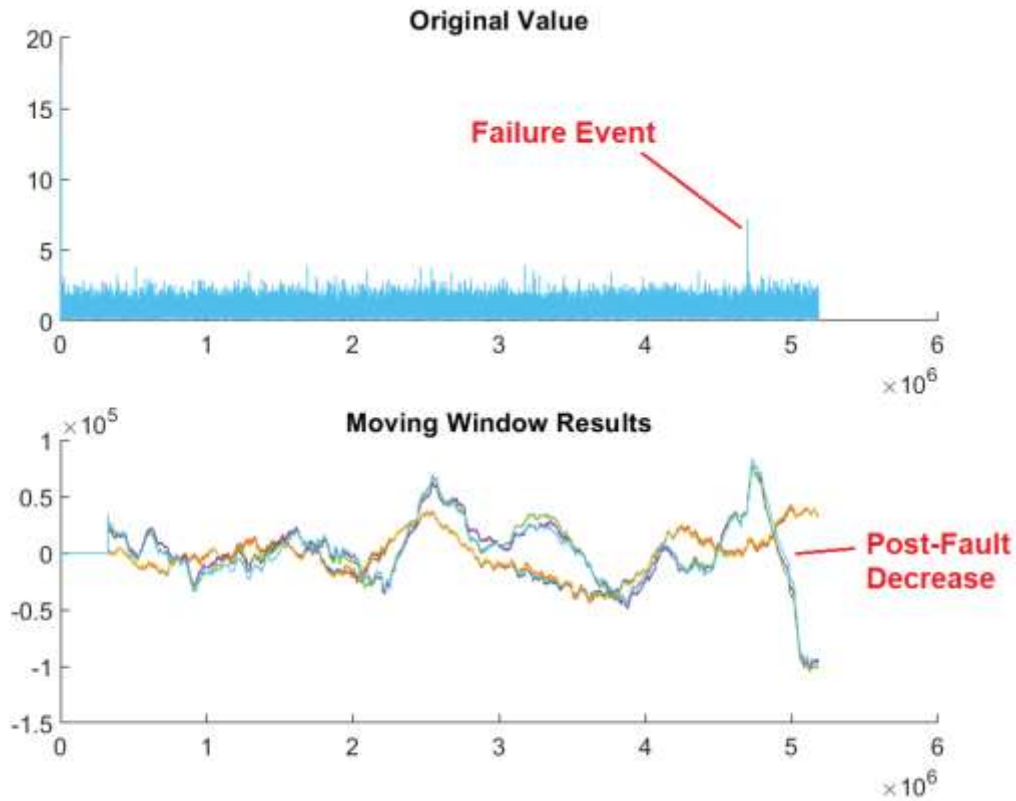


Figure 2.12 More subtle trends extracted from bushing failure data by the sum of Z-scores method.

In cases where SZS trends are clearly steeply increasing, the value obtained from SZS analysis may be directly employed in assessing probable asset health in the vicinity of the measurement site. In cases where SZS trends are more subtle, it may be desirable to use AI and ML techniques, rather than human inspection, to detect probable failure events. This aspect is explored below.

The approach of bifurcating past real-world power system data into *normal*, *alert*, and *alarm* groups and training ML models based on these distinctions was taken in this part of the investigation. Figure 2.13 depicts how data from real power system events was preprocessed, classified, and used in training and model testing. The ML model chosen for this investigation was the Support Vector Classifier (SVC), a well-characterized and mature algorithm available as a pre-designed package in the Python library *scikit-learn*.

In particular, the following steps were taken to initially train and subsequently test the SVC models produced:

1. Approximately 18 hours of data prior to an asset failure was extracted from a PMU data archive (this value was selected based on previous plots of SNR standard deviation and SZS, which showed this to be the period of time during which asset health degradation was most clearly evident)

2. The extracted data was preprocessed using SNR standard deviation conversion and the SZS method
3. The data was manually labeled according to its temporal proximity to the asset failure event: the samples from the 0-6 hour period before the event was classified as "alarm", the samples from the 6-12 hour period before the event were classified as "alert", and the samples from the 12-18 hour period before the event was classified as "normal"
4. The labeled data set was split randomly into training and testing data: 70% of the samples were used for training the SVC model and 30% of the samples were reserved for model testing later on. Importantly, no training data was used for testing the model and no testing data was used during the training
5. The SVC model was trained on the labeled training data
6. The trained SVC model was then provided with unlabeled randomized testing data and was allowed to make predictions about the state of health ("normal", "alert", or "alarm") of the power system at each sample
7. The assigned labels of the testing data, stored separately from the data provided to the SVC model, were compared against the model predictions in order to establish the model accuracy as a percentage

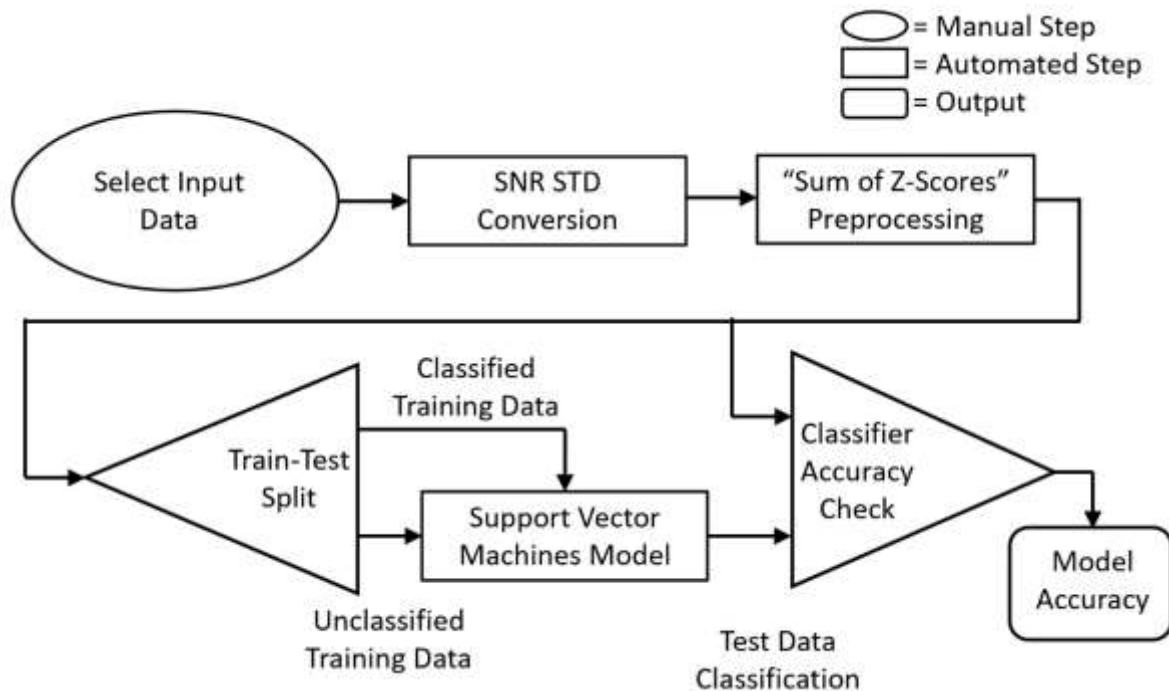


Figure 2.13 Flowchart of the SVM algorithm training and evaluation methodology.

The SVC is an extension of the Support Vector Machine (SVM), and functions as a classifier by mapping a non-linear separation boundary between classified data points in a multi-dimensional space of features [6]. Although there are a number of different ways to perform this mapping, one commonly-used technique for implementing it is using a radial basis function (RBF) kernel that maps the relationships between the training data points in n -dimensional feature space to a higher-

dimensional (or infinite) feature space as a function of their distance from one another. This is expressed mathematically as follows:

$$K(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2} \quad (5)$$

where $K(x_1, x_2)$ is the value of the RBF kernel function for a given set of related points x_1 and x_2 , γ is the distance scaling factor, and the operation $\| \cdot \|$ denotes the Euclidean distance between the two related points in multi-dimensional space.

There are two hyperparameters of the SVM model with RBF kernel: γ and C . γ scales the relative squared distances between the data points, and C sets the penalty for misclassified data points during the SVM training [6]. Several values of hyperparameters were tested before a final selection was made on the preferred hyperparameters to be used in this model.

Figure 2.14 shows the preprocessed SNR standard deviation data for the lightning arrester failure event alongside its manually assigned labeling. On the top classification plot, a value of 0 represents a "normal" classification, a value of 1 represents an "alert" classification, and a value of 2 represents an "alarm" classification.

Following the procedure outlined by the flow diagram of Figure 2.13, this labeled data set was split into training and testing data to train an SVC model, and finally the accuracy of that model was evaluated.

The input data and output accuracy metrics for the predictions made by the trained SVC model are summarized in Figure 2.15. Of the 19,440 fragments of feature data provided to the SVC model for evaluation, only 638 were misclassified. For this event, the overall accuracy was thus 96.7% for the trained SVC model using the RBF kernel and hyperparameters selected.

This high accuracy suggests that, for this particular event, the SVC model operating on SZS-preprocessed data is a good choice for asset health prediction. In order to establish if this methodology was also more generalizable, it was subsequently tested on an additional power system failure event as described below.

In order to provide further evidence of the suitability of the SVC model trained on labeled SZS data, a second real power system event was selected for additional evaluation of the process described. This event was a transformer bushing failure, and the data employed in training and testing the model was the same data that produced subtle, non-obvious SZS changes in the plot of Figure 2.12.

The same procedure of preprocessing, training, and testing the SVC model as was performed in the lightning arrester failure event was now performed on the bushing failure event data. Upon completion of the training and testing steps, the accuracy results summarized in Figure 2.16 were obtained.

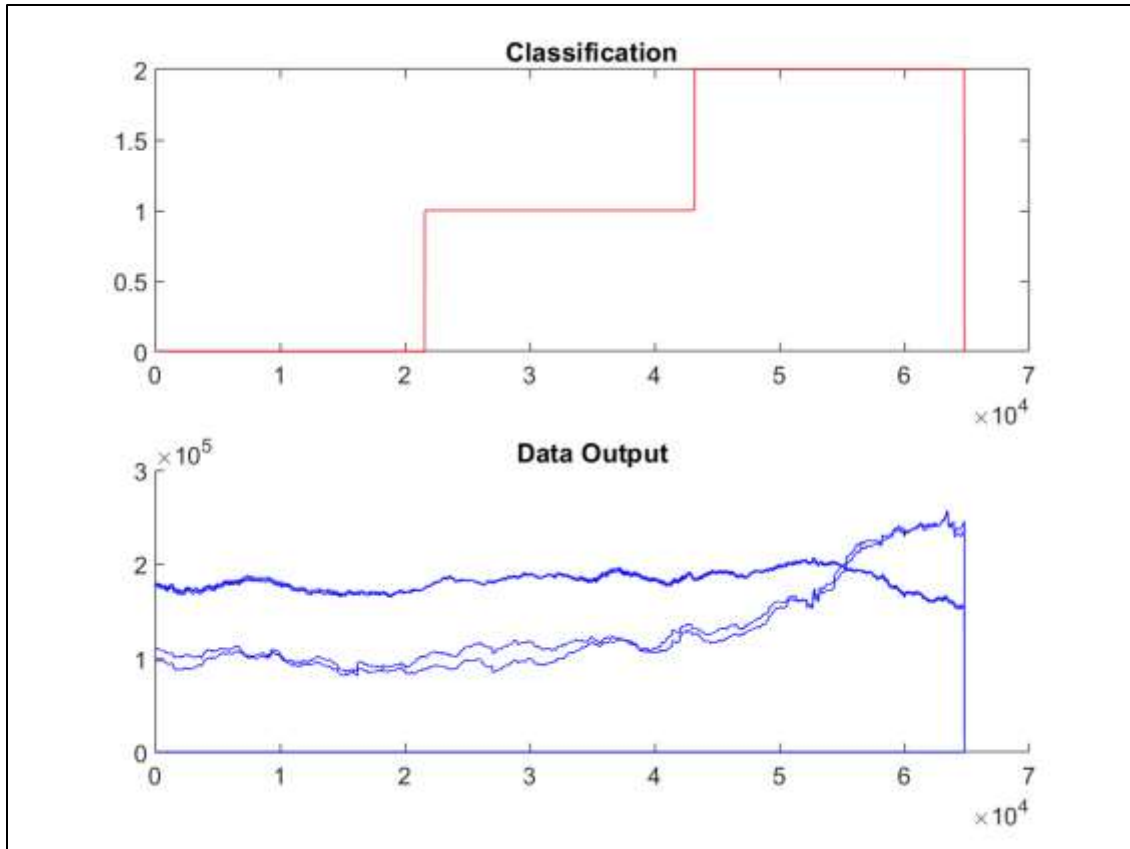


Figure 2.14 Timestamp-based labeling of input training and testing data.

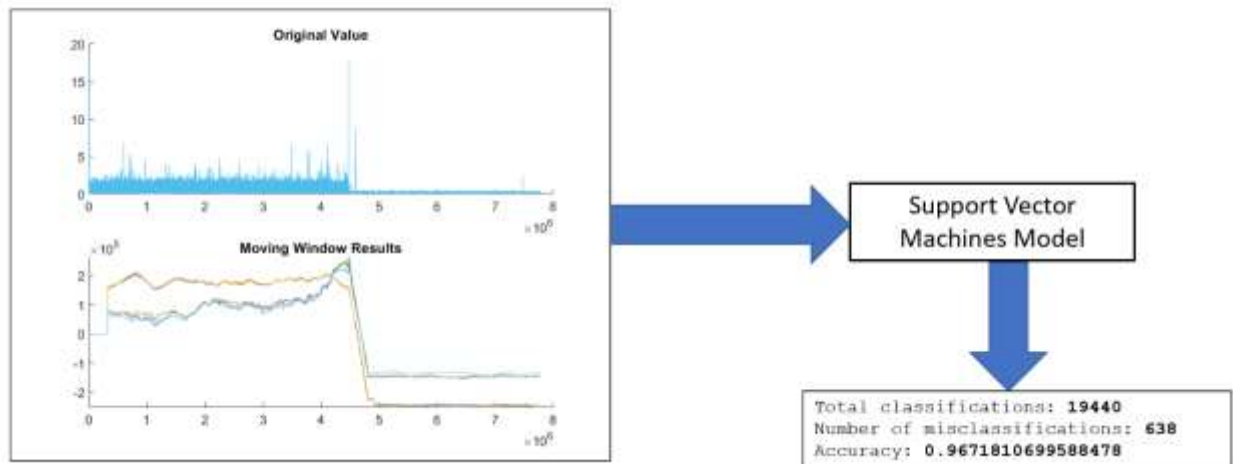


Figure 2.15 Accuracy of SVM model on 30% randomly selected testing data (no training data used for testing) for lightning arrester event.

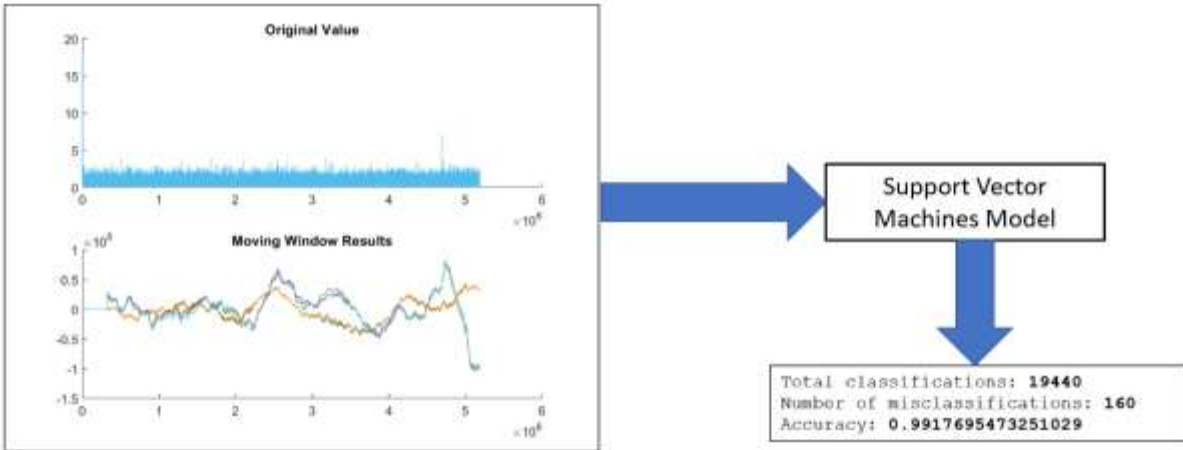


Figure 2.16 Accuracy of SVM model on 30% randomly selected testing data (no training data used for testing) for transformer bushing event.

Here it can be seen that for the same number of samples provided to the trained SVC model, the number of misclassifications was even lower than that obtained for the lightning arrester failure. The overall classification accuracy of this SVC model was over 99%. This provides further evidence to suggest the viability of this process as a method to leverage AI to predict and locate probable asset degradation in the field.

Although further testing will be required to evaluate which types of events this methodology is best suited to assessing, this process was deemed sufficiently promising to warrant the construction of a more scalable, automated set of programs to enable the process to be leveraged on a wider scale. The programs developed for this purpose are discussed in the next section.

2.3 Creation of a Generic Framework for Simultaneous Health Monitoring of Diverse Power System Assets

Following the promising development of an AI-based methodology for predicting power system asset degradation based on PMU data, it was deemed an important step to make this process more streamlined and ready for wider-scale testing. As a result, a generic framework for implementing the SVC classification method with SZS preprocessing was developed and deployed for testing at a local electric power utility (who is also a PSERC member).

In particular, a set of Python scripts capable of being run in almost any modern computing environment were developed. The scripts are provided in Appendices A-C. The first script, "SNR_STD_exporter.py," serves as the initial preprocessor for ingesting and preparing data for SZS and ML evaluation (see Appendix A). As depicted in Figure 2.17, "SNR_STD_exporter.py" allows the end user to locate a source file of PMU data (preferably in comma-separated value, or CSV, format). It additionally allows the user to specify the CSV column number associated with each power system measurement of interest, including:

- A-phase voltage magnitude
- B-phase voltage magnitude

- C-phase voltage magnitude
- A-phase current magnitude
- A-phase current magnitude
- A-phase current magnitude

Execution of "SNR_STD_exporter.py" not only converts the input CSV data to SNR standard deviation data for immediate use in the subsequent model-building script, but also provides human-viewable previews of each of the time-series data sets produced, including both the input data plots and output SNR standard deviation plots (as shown in Figure 2.17).

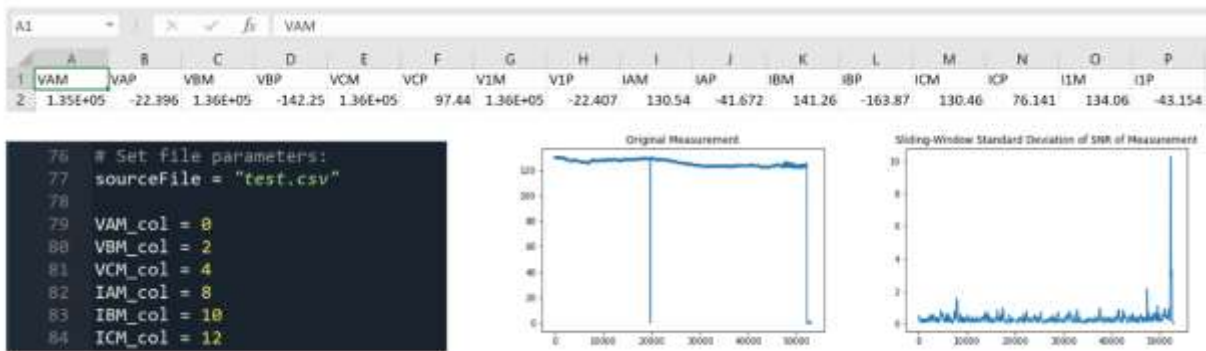


Figure 2.17 Features and outputs of "SNR_STD_exporter.py".

The second script available for use in the generic framework for asset health monitoring is called "SZS_validate.py," and its purpose is to ingest preprocessed SNR standard deviation data, perform the SZS method on it, and finally generate and validate an SVC model based on the input data (see Appendix B). "SZS_validate.py" not only produces a pre-trained SVC classifier model that can be re-imported into other Python scripts for subsequent use, but also it provides an internal validation of this model on provided data in order to ensure that the mode is behaving correctly. In Figure 2.18, one instance of the SVC model is trained on a small amount of toy data and its accuracy is shown. In addition to the returned model file (shown in Figure 2.18 as "trained_model.svm"), the Python script "SZS_validate.py" also provides a human-viewable plot of the SZS values obtained, and additionally provides a detailed sample-by-sample spreadsheet of the validation test predictions and true values in order to manually fine-tune the model to provide optimal results.

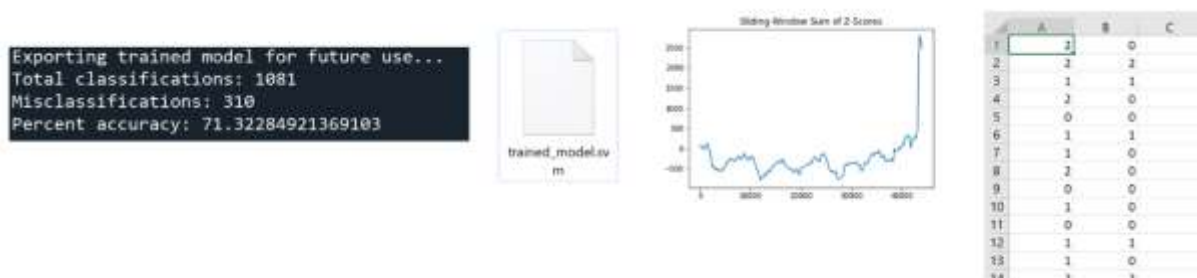


Figure 2.18 Features and outputs of "SZS_validate.py".

The third Python script provided for use in this generic framework for asset health monitoring is called "SZS_classify.py" (see Appendix C). This script is designed to read in an already-trained SVM model (trained by "SZS_validate.py") and to use this trained model to make predictions about new, never-before seen data.

Once a model has been trained on the behavior of a particular region of the power system, it is intended to be used on any incoming data needing to be assessed for asset degradation indications for that region. By providing preprocessed data and a trained model to "SZS_classify.py," classifications are made automatically on the provided data, without any need for manual labeling. Figure 2.19 depicts the outputs of "SZS_classify.py" when provided with toy data for evaluation. Plots of the SZS of the data, the sample-by-sample classification of the data, and most importantly, a spreadsheet containing the sample-by-sample predictions made by the SVC algorithm are written to disk once execution is complete.

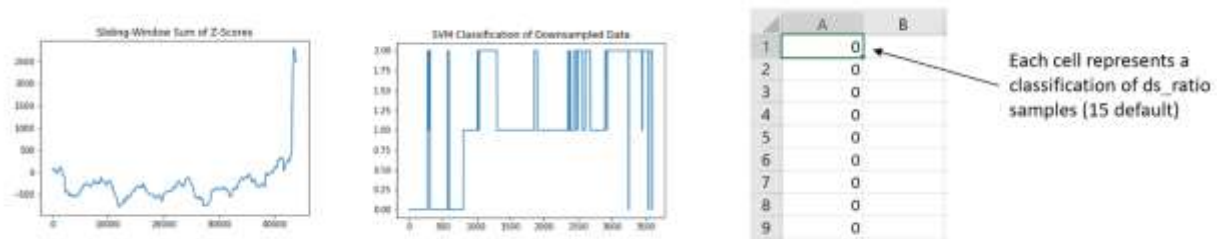


Figure 2.19 Features and outputs of "SZS_classify.py".

The above scripts were provided to a local electric power utility for their own in-house evaluation on real-world power system data. Results of the evaluation are currently pending. It is important to note that the SVC-based framework proposed here works best when trained on data acquired from the region of the power system where it is intended to be deployed. Because the latent noise level of different areas of the power system may vary considerably, even before and after an event as shown in Figure 2.20, localized training is advantageous to producing good model accuracy.

In light of this, it is important to note that not all areas of the power system will have readily available fault-state data sets. Thankfully, power system asset failures are not particularly common; however, this results in a dearth of abnormal-state training data. This is an ongoing topic of research: the ability to produce accurate synthetic data mimicking the fault behavior of particular parts of the power system is crucial to enabling the wider-scale deployment of ML-based anomaly detection schemes such as the one proposed.

In the interest of pursuing an anomaly detection method that does not rely as heavily on labeled training data, another technique investigated in this section was the Matrix Profile technique. Developed by Eamonn Keogh at the University of California Riverside and Abdullah Mueen at the University of New Mexico [<https://towardsdatascience.com/introduction-to-matrix-profiles-5568f3375d90>], this method works by identifying repeating trends in time-series data (referred to in Matrix Profile parlance as "motifs"), and subsequently identifying anomalous deviations from these trends (referred to in Matrix Profile parlance as "discords").

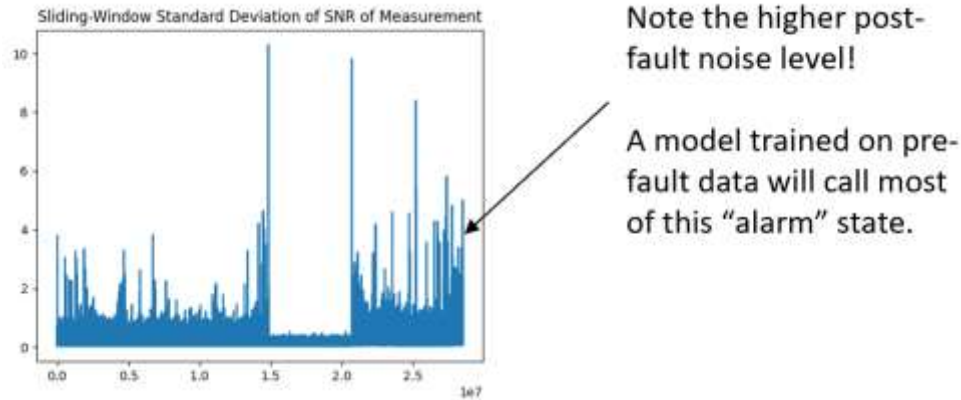


Figure 2.20 Changes in the SNR STD profile after a fault due to changing system operating point.

In order to test a practical implementation of Matrix Profile, the *stumpy* Matrix Profile Python library developed by TD Ameritrade was employed and tested on data previously investigated using the methods already discussed.

Despite the possibility that Matrix Profile would provide an unsupervised anomaly detection methodology, there was not enough repeatability in SNR standard deviation data to allow Matrix Profile to effectively identify pre-fault anomalous data as a discord, even under several different window size settings and data down-sampling ratios. Examples of the ambiguous results provided by Matrix Profile on real power system data can be seen in Figure 2.21. Although a significant increase in the motif distance did occur in the case of a Matrix Profile window size of 30000, this change returned to baseline levels *before* the asset failure occurred; thus, it was not considered a strong indicator of probable asset degradation. A further reason to avoid Matrix Profile for this particular application was its very long processing time, which was found to be several times slower for a given size data set than the SZS and SVC methods.

2.4 Completion of a Generic Framework for Simultaneous Multiple-Substation Health Monitoring of Diverse Power System Assets

With the exception of the multi-source limit counting strategy, all of the above-described methodologies for asset health assessment are designed to be operated on one data source at a time, for example, the data streamed from one PMU in one substation. The overall goal of this portion of the project is to investigate a method of asset health monitoring in multiple substations simultaneously. As such, an additional methodology was investigated with the goal to make it:

- Fast enough to operate on a massively parallel scale on real-time data without excessive computational burden
- Capable of simultaneously incorporating many data sources into a single power system health assessment
- Able to operate unsupervised without the need for localized training

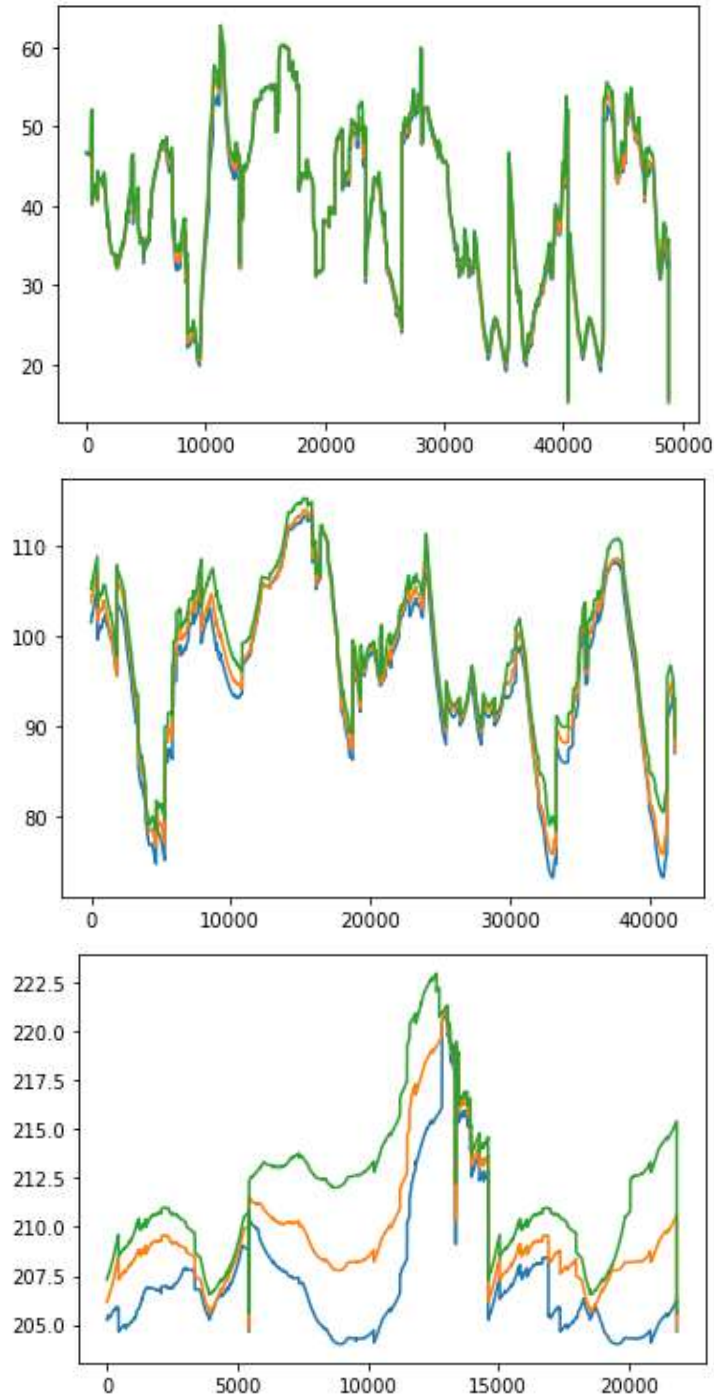


Figure 2.21 Testing of Matrix Profile for detecting anomalies in periodically varying SNR STD data on 48 hours' data for $W=3000$, $W=10000$, and $W=30000$. Vertical axis represents distance between motifs and horizontal axis is time-series sample number

With these goals in mind, a separate approach was taken to assess system anomalies that altogether bypasses the need for SNR standard deviation analysis, Z-score summation, and model training. This approach, referred to as the *derivative difference* method, operates on multiple simultaneous data sources by:

1. Time-synchronizing all available data samples
2. Measuring the *rates of change* or derivatives of each data set over time
3. Subtracting the derivatives from *multiple data sources* to localize regions of the power system with *elevated rates of evolution*

The process by which this methodology can be applied to power system data is summarized in Figure 2.22.

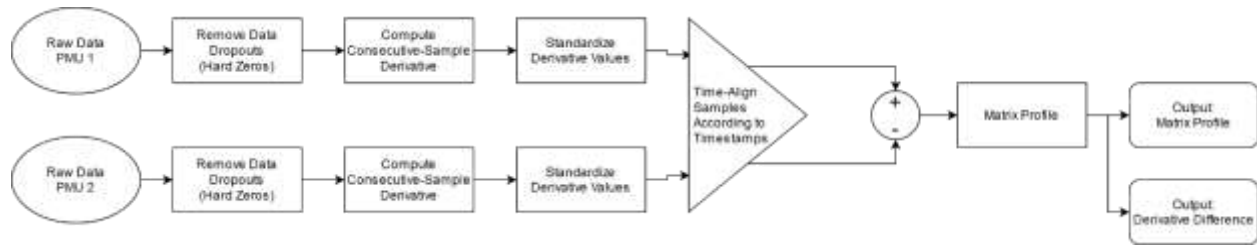


Figure 2.22 Flow diagram of derivative difference method

The derivative difference method in action can be seen in the multi-step process depicted in Figure 2.23. Consecutive samples X_{n-1} and X_n are subtracted for each time-series data source, the resultant difference is averaged, and the end result is a plot that compares the rates of change of one area of the power system to another. Prior to subtraction, derivative values may also be statistically normalized to the mean and standard deviation of their respective sources in order to allow all derivatives to be compared to one another at a common scale.

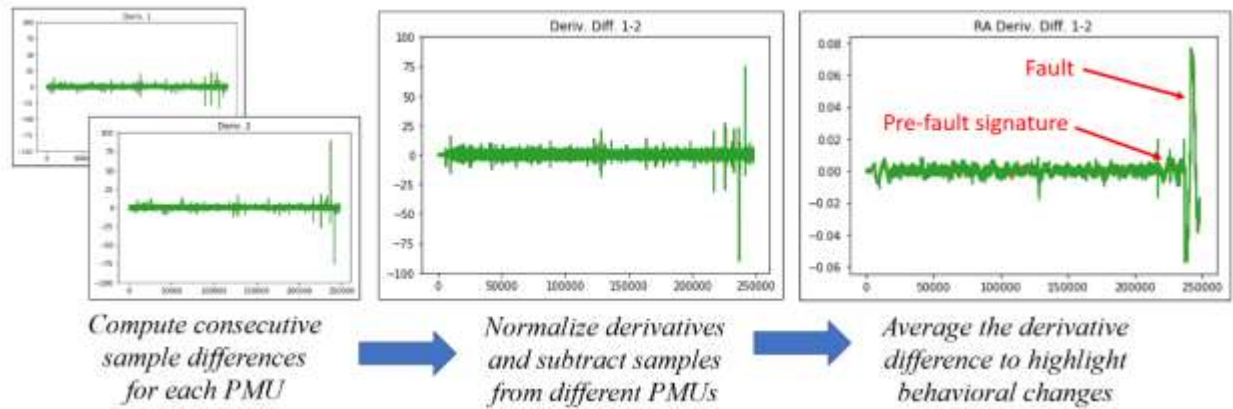


Figure 2.23 Derivative difference method in action on real power system data

The advantages of such a method become clear when even more data sources are incorporated into the analysis. When a large number of power system measurement sites are compared against one

another, a "heatmap" matrix of relative rates of evolution is obtained. Regions of this heatmap with greater rates of system evolution are more likely to contain sources of anomalous system behavior than regions which are relatively stable as compared to their neighbors. A small-scale example of such a matrix is shown in Figure 2.24.

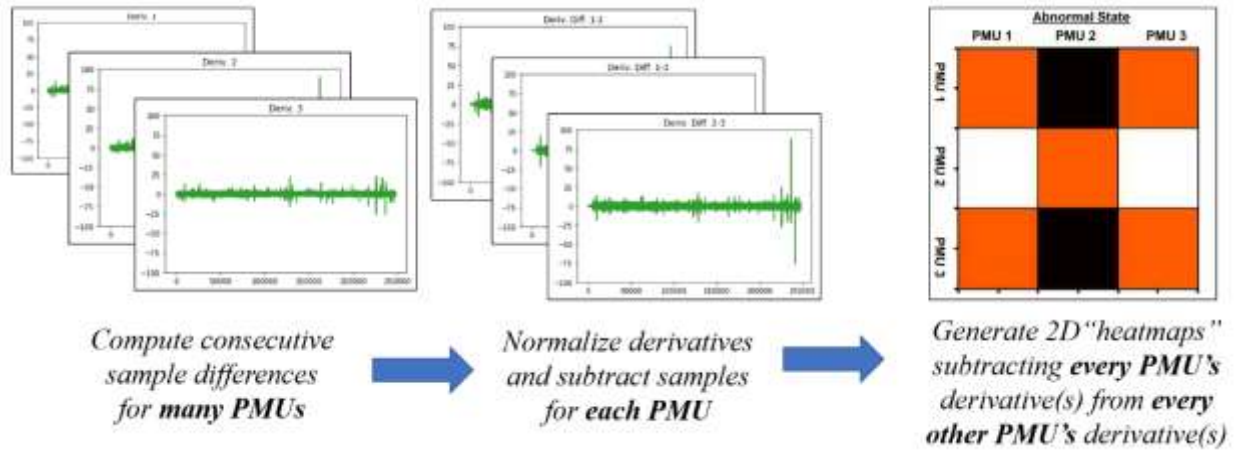


Figure 2.24 Multiple derivative differences combined into a series of 2D "heatmap" frames

Because a new "heatmap" matrix is produced for every consecutive sample of incoming data from the power system, it is possible to create a real-time frame-by-frame video of these matrices as they evolve. Figure 2.25 depicts the evolution of the matrix shown above in Figure 2.24 as the normal background behavior gradually evolves into abnormal behavior. It is clear that a change has taken place, and this change could be observed by a power system operator or even an automatic monitoring algorithm in real-time.

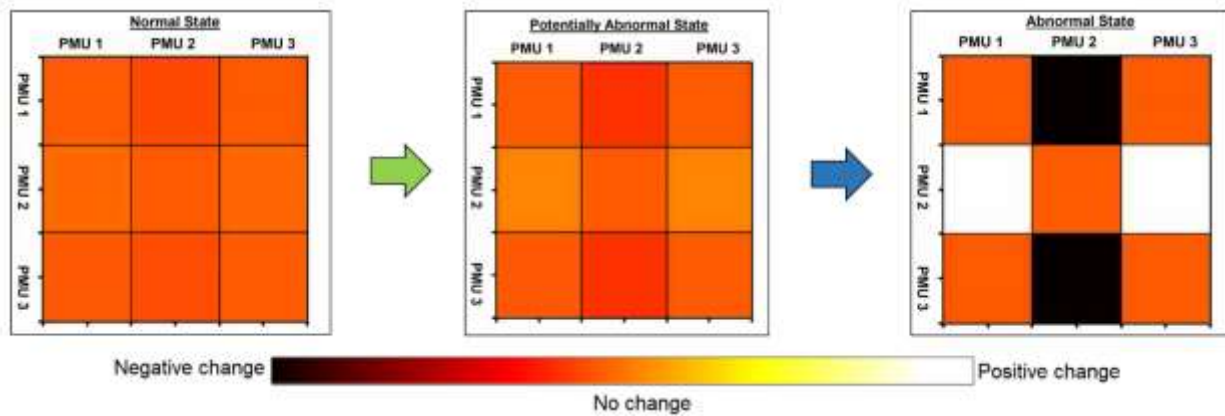


Figure 2.25 Evolution of heatmap frames prior to catastrophic asset failure

One final note pertaining to the value offered by this method as compared to all the previously-discussed methods is its relatively high computational speed and low computational burden. Because only a few simple mathematical operations must be performed on each incoming sample in order to obtain its derivative and normalize it, this process can easily be performed on large numbers of parallel data streams.

The most computationally burdensome step in this type of analysis is the computation of the derivative difference heatmap matrix itself, as it requires N^2 subtractions of the incoming data streams to be performed. Nevertheless, for typical power systems with hundreds or even thousands of measurement sources, this calculation would still be trivial in modern computer hardware with multiple processor cores operating at gigahertz speeds [7].

2.5 Future Work: Automation of Event Detection in Simultaneous Multiple-Substation Health Monitoring of Diverse Power System Assets

The methods for simultaneous multiple-substation health monitoring of diverse power system assets discussed here effectively leverage artificial intelligence methods to distill specific figures of merit about the health of regions of the power system from the vast quantity of data supplied by PMUs. Despite the automation inherent to these processes, they all require some final decision to be made by a power system operator as to whether the *signs of failure* that these algorithms report are sufficiently strong to warrant real-world deployment of corrective measures.

A significant research opportunity still exists to develop methods to leverage AI to make *operational decisions* based on the real-time results supplied by these algorithms. In particular, the highly scalable two-dimensional matrices generated by the derivative-difference method are potentially well-suited to decision-making algorithms based on convolutional neural networks (CNNs), which have traditionally been leveraged to perform image processing and image classification, both of which are performed on similar two-dimensional data.

Figure 2.26 gives a summary of how a derivative difference matrix might be used to train an ML algorithm to make operational decisions about where and how to deploy crew to monitor or diagnose equipment failure.

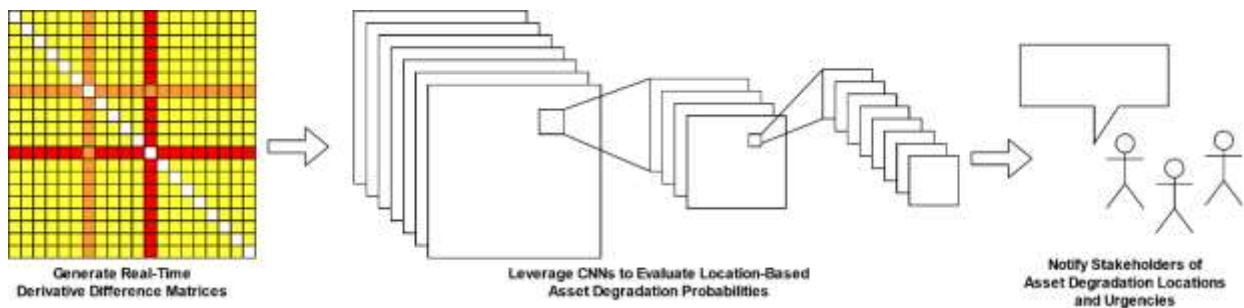


Figure 2.26 Concept of ML-based operational decision-making based on derivative difference heatmap data

Another opportunity lies in Generative Adversarial Networks (GANs). GANs have been used in anomaly detection frameworks due to their ability to generate realistic data. This generation of data (also known as data augmentation) mitigates the problem of highly biased dataset of normal conditions as opposed to abnormal conditions inherent to asset health problems. In this setting, a GAN can be developed and trained to generate abnormal data. A standard architecture for GAN is shown below along with the generated data.

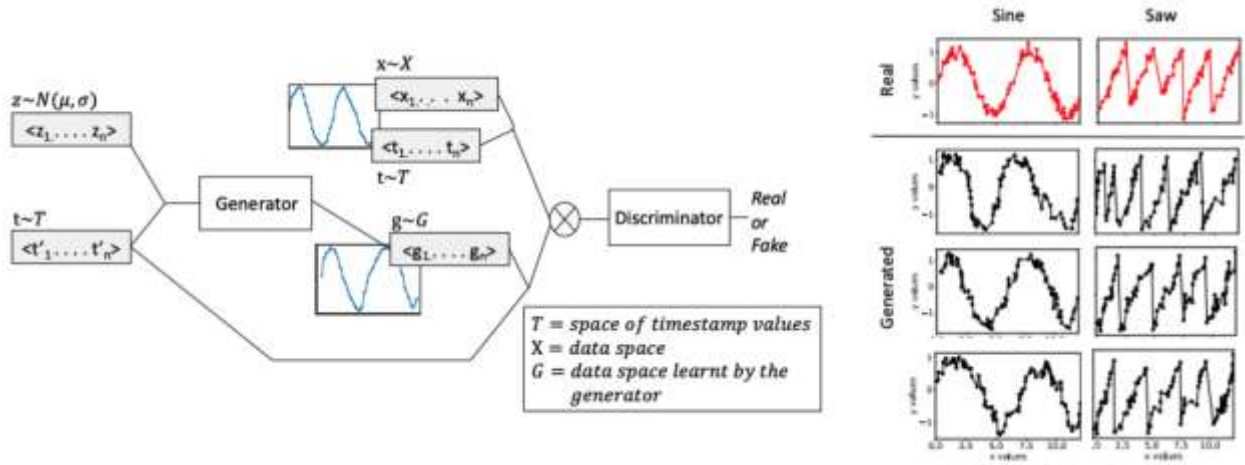


Figure 2.27 On the left is a simple GAN model and on the right are time-series generated samples [8]

This newly generated dataset of normal and abnormal data can then be turned into a classification problem where the time series is passed through a normal NN, CNN, LSTM (Long short-term memory) or GAN. Those networks have been thoroughly investigated when it comes to classification problems and have worked well in the past.

In [9], a GAN based approach was used to detect pointwise and window-based anomalies in time-series. It shows how GAN and reconstruction-based methods are better in anomaly detection compared to prediction methods, e.g., LSTM, and classical statistical methods, e.g., ARIMA. Similar models can be used or even a combination of classical and ML-based models can be used to improve performance.

3. Deep Learning Approach to Power System State Estimation and Bad-Data Detection

3.1 Introduction

This report focuses on achieving situational awareness in power systems with limited measurements. Specifically, we are interested in power system state estimation and anomaly detection when the system is unobservable due to the lack of sensor measurements. This work has been motivated by the greater presence of distributed energy resources, especially in the form of inverter and converter-based resources that make the power grid increasingly stochastic, dynamic, and vulnerable to system/cyber anomalies. To this end, this report covers two aspects of situational awareness: (i) state estimation of an unobservable distribution system and (ii) detection of anomaly sequence in high-resolution measurements.

3.1.1 Bayesian State Estimation of Unobservable Distribution System via Deep Learning

The state estimation problem is motivated by the need of coping with the rising presence of distributed energy resources (DER) in distribution systems. To unlock the full potential of DER, effective state estimation is necessary for distribution systems to provide tighter control of power flow for real-time operations.

Unlike the transmission systems where redundant measurements are collected by a supervisory control and data acquisition (SCADA) system, current distribution systems have few measurements such that the system is generally *unobservable* [10],[11]. By unobservable it means, mathematically, that the measurement function is not invertible, and there is a manifold of uncountably many states that correspond to the same measurement. Consequently, classical techniques such as the weighted least squares (WLS) method that assumes deterministic states, if applied blindly, fail in practice.

A standard remedy of unobservability is to use the so-called pseudo-measurements based on interpolated measurements or historical data. These techniques are ad hoc and suboptimal. Additional measurements may come from smart meters at the edge of the network, but these measurements arrive at a much slower time scale, *e.g.*, in 15 minutes to one-hour intervals. Such type of data is incompatible with the more rapid changes of DER such as solar generations and difficult to use to provide minute or sub-minute level situation awareness. Realizing state estimation for real-time operations, therefore, requires a fundamentally different approach from the state of the art, one that overcomes the lack of observability.

State estimation for unobservable systems must incorporate additional properties beyond the measurement model defined by the power flow equations. To this end, we pursue a *Bayesian inference* approach where the system states (voltage phasors) and measurements are modeled as random variables endowed with (unknown) joint probability distributions. Given the highly stochastic nature of the renewable injections, such a Bayesian model is both natural and appropriate.

The most important benefit of Bayesian inference is that observability is no longer required. A Bayesian estimator exploits the probabilistic dependencies of the measurements on the system states, and it improves the prior distribution of states using available measurements, even if there are only a few such measurements that render the system unobservable.

The advantage of Bayesian inference, however, comes with several nontrivial difficulties in applications. First, the underlying joint distribution of states and measurements, if unknown, needs to be learned. Second, even if the probability distribution of the states and measurements is known or learned, computing the actual state estimate is often analytically intractable and computationally prohibitive.

3.1.1.1 Related Work

State estimation based on deterministic models of states has been extensively studied. See [10],[11] and references therein. Existing Bayesian techniques that model states as random variables are less common even though the idea was already proposed in the seminal work of Schweppe [12]. Bayesian methods for state estimation can be classified into two categories: Bayesian pseudo-measurements [13-16] and Bayesian state estimation [17-24]; the former uses probability distributions to generate pseudo-measurements so that conventional techniques such as WLS can be applied. Such hybrid techniques are suboptimal but can leverage the existing state estimation methods. The latter "direct" Bayesian solutions use distribution information explicitly and minimize the mean squared error. These techniques vary in how the estimator is computed. The method proposed in this project falls into this latter category.

Direct Bayesian state estimation requires the computation of the conditional mean of the state variables. One approach is based on a graphical model of the distribution system from which belief propagation techniques are used to generate state estimates [17,18]. These techniques require a dependency graph of the system states and explicit forms of probability distributions. Another approach is based on a linear approximation of the AC power flow [19]. The proposed approach belongs to the class of Monte Carlo techniques in which samples are generated and empirical conditional means are computed. In our approach, instead of using Monte Carlo sampling to compute the conditional mean directly as in [20], Monte Carlo sampling is used to train a neural network that, in real-time, computes the MMSE estimate directly from the measurements.

Neural networks have been proposed for state estimation as early as 1991 in [21]. Different architectures of neural networks have been considered: an auto-encoder structure in [22] and a radial basis function network in [23]. Although it is not cast as a Bayesian state estimation, the approach in [24] appears to be quite close to ours where a neural network with two hidden layers was used to estimate states directly using load bus measurements as the input of the neural network. The distributions of power injections were not learned but assumed in [24], and issues of bad data were not considered.

Bad data detection and identification has been studied extensively. See [10],[11], and references therein. Classical methods are *post state estimation* techniques where states are first estimated and used to compute the residue error. The presence of bad data is declared if the norm of residue error exceeds a certain threshold. For such techniques, system observability is a prerequisite. Even when

there are enough measurements such that the system is observable, to identify and remove bad data, an iterative process is often used where state estimation is performed repeatedly after each positive bad data detection and removal. Such techniques often fail to identify bad data and mistakenly remove good data. Indeed, there is no general method to detect, identify, and remove bad data effectively [11]. There seems to be limited work on the use of the Bayesian approach to bad data detection and cleansing.

3.1.1.2 Summary of Results and Contributions

The main contribution of this work is a novel application of Bayesian state estimation with bad data detection and cleansing for possibly unobservable systems or one whose measurements are highly unreliable, corrupted by bad data, or subject to data attack. We propose a machine learning approach for the minimum mean squared error (MMSE) estimation of system states and a Bayesian bad data detection and cleansing technique to identify and remove bad data.

The proposed machine learning approach consists of *distribution learning* and *deep regression learning*; the former uses smart meter data to learn bus injection distributions from which training samples are drawn to train the neural network in regression learning. This particular architecture is not necessarily new in general, but it appears to be novel for state estimation and bad data detection problems in power systems. A key innovation here is a way that the physical power system model (the power flow equations) is embedded in the regression learning. Another innovation is the learning of fast time-scale power-injection distribution from the slow time-scale smart meter data.

Besides overcoming system unobservability, the proposed neural network approach offers two additional advantages: First, the proposed Bayesian approach provides a direct and simple way to identify and remove multiple bad data measurements, which is known to be intractable for conventional (WLS-based) bad data solutions. Second, the neural network approach to MMSE estimation has a substantial advantage in real-time computation over the WLS-based methods.

Numerical results demonstrate several features of the proposed approach. First, we show that the proposed Bayesian state estimator performs considerably better than the benchmark pseudo-measurement techniques, including the ones that generate pseudo-measurements using neural networks. Second, we show that the proposed technique provides a level of robustness against inaccuracies in distribution learning. In particular, the Bayesian state estimator performs well when the distribution is learned from samples drawn from one distribution and tested on data from different distributions¹. Third, our results suggest that using deep learning seems essential. Fixing the total number of neurons and varying the number of layers in the neural network, we observe that neural networks with 5 layers or so provide better performance, and networks with bulging middle section are better than rectangular networks. Finally, simulations show that the Bayesian bad data detection and cleansing is considerably more effective than the non-Bayesian pseudo-measurement techniques.

¹ In our simulation, we generate test data based on a distribution learned in a different year.

The proposed technique is of course not without limitations. Some of these limitations are summarized in Section 3.3.6.

3.1.2 A Deep Learning Approach to Anomaly Sequence Detection

We consider the problem of detecting data and system anomalies using high-resolution power system measurements. In particular, we consider phasor measurements at reporting rates up to 256 Hz and continuous point-on-wave (CPOW) measurements sampled at up to 100 kHz. At these sampling rates, power system measurements exhibit strong temporal dependencies.

High-resolution measurements currently exist in the field in various monitoring devices [25], often for post-event analysis and rarely streamed to the control center for real-time monitoring. However, with the increasing penetration of inverter-based resources, there are cogent needs for high-resolution wide-area monitoring that goes beyond using low-resolution SCADA and PMU based measurements [25,27]. To this end, we aim to fill a theoretical and practical gap in using high-resolution measurements to detect anomalies at the sensor level and combine sensor-level detections at the control center to form more accurate anomaly detections.

The challenge of anomaly detection with high-resolution measurements is threefold. First, temporal dependencies in high-resolution measurements are difficult to model. Conventional techniques based on sample-by-sample detection or assuming independent data samples tend to perform poorly. In this work, we stress the significance of *anomaly sequence detection* where anomaly detection is made based on a measurement sequence rather than individual samples.

Second, anomalies are rare events, and there are uncountably many possibilities that anomalies can occur; no single model nor sufficient historical data are available to characterize and validate anomaly data. Therefore, anomaly detection should be derived from the anomaly-free model only, independent of the types of anomalies that may occur.

Finally, defining "normalcy" is nontrivial. While a power system has well-defined nominal operating conditions, it has frequent transients as generators are dispatched in real-time. There is no standard data model that leads to well-defined statistical tests for anomaly-free data.

We consider three types of anomalies. One is the conventional bad data caused by malfunctioning sensors and communication errors that generate outliers. The second is data anomaly from data attacks, where an adversary manipulates sensor data to affect the operator's decision process. The third is system anomalies such as faults and operation contingencies that manifest themselves in data. We do not distinguish among different data anomalies.

3.1.2.1 Related Work

The classic anomaly detection in wide-area power system monitoring is bad-data detection in the context of power system state estimation [10,12,28]. A standard approach is *post-estimation bad-data detections* where state estimation is performed first as if there were no bad data. Data anomaly is declared when the (normalized) residue error (computed using the estimated state) is greater

than a certain threshold. As a result, inaccuracy of state estimation caused by anomaly data circulates back to affect bad-data detection.

An alternative is the *pre-estimation bad-data detection* techniques that detect data anomaly before state estimation, thus breaking the path of estimation error propagation. The key idea is to replace the estimated state in the post-estimation scheme with a *predicted state* using the past measurements and apply residue test on the predicted measurement [29,30]. Such techniques exploit temporal dependencies in the data for prediction, thus more relevant to the anomaly sequence detection problem considered in this project. These techniques, however, assume specific temporal dependency models that are difficult to obtain.

A more direct pre-estimation approach is to detect anomalies based on features of anomaly-free data. One of the earliest such techniques is using a neural network classifier trained with anomaly-free data [31]. A separate line of approaches is to extract features from the anomaly-free data and classify data in the feature space. Examples include the use of principal component analysis to characterize the signal subspace of the anomaly-free data [32] and the formulation of the problem as the detection of a change in measurement probability distribution [33].

There is a growing literature on the use of machine learning for bad-data detection in power systems since the mid-1990s [31]. These techniques can be categorized based on how data are used in learning. Supervised learning requires the labeled training data in the anomaly-free and anomaly cases [34-36], semi-supervised learning requires training samples for the anomaly-free data [37-39], and unsupervised learning requires no training data [33,40]. An ensemble learning technique is proposed in [41] that combines a collection of bad-data detectors.

Because it is difficult to obtain labeled anomaly data for training, the semi-supervised and unsupervised learning paradigms are of particular significance. Although not designed for power system state estimation, two types of semi-supervised anomaly detectors that use only training samples under the anomaly-free model can be applied for bad-data detection in power systems. One is the one-class support vector machine OC-SVM [42] that separates anomaly and anomaly-free data deterministically. The other is based on the idea of auto-encoder in deep neural network [43]. An implicit assumption is that anomaly and anomaly-free data do not share a common data domain for these methods, which rarely holds in power system measurement models.

Statistical learning approaches to anomaly detection start from the premise that anomaly and anomaly-free data come from different probability distributions. To this end, a recent work of particular relevance is [37] that focuses on dynamic data attacks of power system state estimation. Although the attack models in [37] suggests an anomaly sequence detection problem, the proposed mitigation strategy is a sample-by-sample detection scheme based on anomaly-free probability distributions from historical samples. The idea of universal bad-data detection methods developed in [38,39] is a semi-supervised learning technique that learns the inverse generative model of the anomaly-free data using a generative adversarial network (GAN) approach using Wasserstein distance [44], followed by a coincidence test. The approach developed in [38,39] relies on that the observations are i.i.d., which is unreasonable for high-resolution data.

There is significant literature on detecting the so-called *data injection attacks* by an adversary who can inject, remove, and substitute data to affect system and market operations [45-50]. In particular, an attacker may create a fake sequence of system states such that the manipulated measurements and the fake state sequence satisfy the underlying power flow equation, which makes the manipulated data *unobservable*. There is no effective anomaly detection solution for such attacks in the literature. The technique proposed here gives a viable solution.

3.1.2.2 Summary of Approach and Contributions

We develop a data-driven machine learning technique to detect anomalies from high-resolution CPOW and PMU measurements. By stressing the significance of *anomaly sequence detection*, the proposed approach is a notable departure from the conventional sample-by-sample detection solutions, and it is perhaps the first anomaly sequence detection method for power system monitoring.

The main technical contribution of this work is twofold. First, we develop a sensor-level non-parametric anomaly sequence detection method in which no assumptions are made for the anomaly data model. The anomaly-free model is also assumed to be unknown, except that historical training samples are available, making the proposed technique a data-driven solution. By not assuming any anomaly model, the proposed sensor-level detection applies to bad-data anomalies, data injection attacks, and system anomalies that manifest themselves in anomaly data patterns. To our best knowledge, there are no existing alternatives in the power system monitoring literature.

A significant challenge of anomaly sequence detection is the unknown temporal dependencies in measurements. To this end, we propose a GAN-based independent component analysis, referred to as ICA-GAN, that transforms anomaly-free measurements with unknown statistical dependencies to uniform independent and identically distributed (*i.i.d.*) samples. We then apply a uniformity test that distinguishes uniform *i.i.d.* samples from the anomaly-free hypothesis from non *i.i.d.* and/or non-uniform anomaly samples. While ICA [51-53] and uniformity tests [54-56] have been developed separately in the past, a combination of them for anomaly-detection is a novel contribution.

Second, we propose a distributed detection framework that combines sensor-level detections for system-level anomaly detection. Such techniques are essential because individual sensors have access to local measurements only, and their detections are likely to be unreliable. Distributed detection plays crucial roles in various surveillance applications and has been studied extensively [57]. Classic techniques require known anomaly and anomaly-free probability models, and measurement samples are assumed to be conditionally *i.i.d.* These assumptions do not apply to the anomaly detection model considered here. To our best knowledge, there are no effective non-parametric decentralized techniques in the literature. The proposed technique exploits a key feature of ICA-GAN that, under the anomaly-free hypothesis, maps dependent measurement sequences to uniform *i.i.d.* sequences, making it possible that local detectors with the same false positive rate (FPR) are used. To this end, we derive a fusion rule that combines individual sensor decisions.

Finally, we test the proposed technique under three anomaly scenarios, using real data set from the EPFL network [58,59] and a larger synthetic Northern Texas network with PMU measurements

[60]. These illustrations cover a natural data anomaly, an unobservable data-injection attack, and a system anomaly. They serve as demonstrations of the versatility of the proposed detection method.

3.2 Bayesian State Estimation via Deep Learning

3.2.1 Bayesian State Estimation and Neural Network

3.2.1.1 Algebraic and Statistical Models

A distribution system is described by a graph $\mathcal{N} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} = \{1, \dots, N\}$ is the set of buses and $\mathcal{E}$ the set of network branches. We assume that node 1 is the slack bus representing the point of common coupling (PCC), where the distribution network is connected to the main grid.

An unbalanced three-phase distribution system is considered. The three-phase voltage phasors at bus i is a complex column vector $x_i = [x_i^1, x_i^2, x_i^3]^T$, where the superscripts are phase indices, and $x_i^k = V_i^k \angle \theta_i^k$, where V_i^k is the voltage magnitude and θ_i^k is the phase angle for the state variable at phase k of bus i . The overall system state $x_i = [x_1, \dots, x_N]^T$ is the column vector consisting of voltage phasors at all buses.

We model the power system by a pair of equations that characterize the relationship among the vector of (complex) power injections $s = [s_1, \dots, s_N]^T$ at network buses, the system states x , the vector of measurements $z = [z_1, \dots, z_M]^T$, and measurement noise e :

$$x = g(s), z = h(x) + e, \quad (3.1)$$

where the system function $g(\cdot)$ is defined by the Kirchoff law and electrical characteristics of the network, whereas the measurement function $h(x)$ is defined by the type and the locations of the meters in the network. Vector z above includes standard types of measurements such as branch power flows, power injections, and current magnitudes. Although the techniques developed here apply to general observation models, we are particularly interested in cases when the system is unobservable, especially for those cases when the number of measurement points is small, often significantly less than the number of state variables. The specific forms and parameters of (3.1) are given in [61].

Aside from the physical models above, Bayesian estimation requires a probability model that specifies the statistical dependencies of the variables. Here we assume that the measurement noise e is independent of the network injection s . Together, the joint distribution of e and s in the product form defines the joint distribution of measurements z and state variables x . The distributions of these random variables are unknown.

3.2.1.2 Bayesian State Estimation

A state estimator $\hat{x}(\cdot)$ is a function of measurement z . In Bayesian estimation, the estimator $\hat{x}(z)$ is defined by the joint distribution of x and z according to a certain objective. We focus on the

minimum mean squared error (MMSE) estimator that minimizes the expected squared estimation error².

From standard estimation theory, the MMSE estimator is given by the mean of the state x conditional on the realization of the observation vector z :

$$\min_{\hat{x}(\cdot)} \mathbb{E}(\|x - \hat{x}(z)\|^2) \Rightarrow \hat{x}^*(z) = \mathbb{E}(x|z). \quad (3.2)$$

Simple as it may appear in the above equation, the computation of the conditional mean can be exceedingly complex. For all practical cases, the functional form of the conditional mean is unavailable. More importantly, perhaps, the underlying joint distribution of x and z are unknown or impossible to specify, which makes the direct computation $\hat{x}^*$ intractable.

3.2.1.3 Neural Network Approximation

Because the Bayesian state estimator is defined on the *unknown* joint distribution of z and x , we consider an alternative approach that learns directly the conditional mean in (3.2) using a deep neural network shown in Figure 3.1. The neural network consists of multiple layers of neurons, and neurons at each layer produce a vector output for the next layer using a (parameterized) nonlinear function of the output vector from the previous layer. The input-output relation of the neural network is a nonlinear map $\mathcal{K}$ such that

$$\hat{x}(z) = \mathcal{K}(w, z), \quad (3.3)$$

where w is the parameter matrix. See [61] for the form of $\mathcal{K}$.

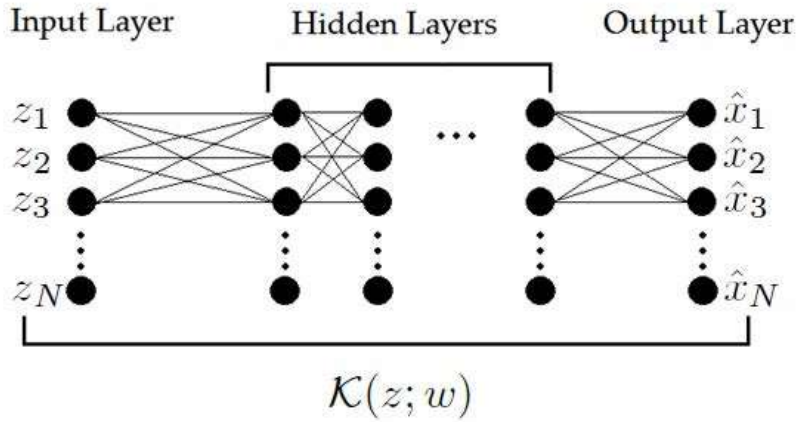


Figure 3.1 Multi-layer Forward Neural Network.

² The developed technique can be also applied to other Bayesian techniques such as the robust estimator based on the minimum absolute error (MMAE) estimator and the maximum a posteriori probability (MAP) estimator.

3.2.2 Learning with Deep Neural Network

3.2.2.1 Training the Neural Network

We now focus on learning the neural network parameter w to approximate the MMSE state estimator $\hat{x}^*(z)$ in (3.2). Standard deep learning algorithms apply to cases when there is a set of training samples made of the input-output pair $\{(z, \hat{x}^*(z))\}$. Such pairs, unfortunately, are not available directly.

The key to obtaining a training data set is to incorporate the underlying physical model characterized by the power flow equation. Although we do not have the joint distribution of x and z , we can draw net injection and noise samples $(s, e) \sim F_s \times F_e$ if we have the marginal distributions F_s and F_e of the injection and noise, respectively. From (s, e) , by the power flow equation (3.1), we obtain (x, z) that can be used for training. Conceptually,

$$(s, e) \sim F_s \times F_e \stackrel{(3.1)}{\rightarrow} (x, z). \quad (3.4)$$

Measurement noise distribution F_e is assumed to be, say, zero-mean Gaussian $\mathcal{N}(0, \sigma^2)$. The distribution of the net power injection, however, depends on a combination of load and possibly renewable generations. We defer the discussion of learning F_s using data from the smart meter and other measurement devices. For now, we assume that the injection and measurement samples can be generated.

Given the set of training samples $\mathcal{S} = \{(x[k], z[k])\}$ generated according to (3.4), the weight matrix w is chosen to minimize the *empirical risk* defined by

$$L(w; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{k: (x[k], z[k]) \in \mathcal{S}} \|x[k] - \mathcal{K}(z[k]; w)\|^2, \\ w^* = \arg \min_w L(w; \mathcal{S}).$$

The empirical risk minimization problem above is well studied for deep learning problems, and an extensive literature exists. See [63] for the use of backpropagation in deep learning. For the state estimation problem at hand, stochastic gradient descent algorithms are considered. In particular, the Adam algorithm [64] designed for non-stationary objectives and noisy measurements appears to be the most appropriate for state estimation problem with measurement noise.

A characteristic of deep learning is over-fitting, which means that the number of neural network parameters tends to be large relative to the training data set. A general approach to overcoming over-fitting is regularization that constraints in some way the search process in neural network training. Standard techniques include L_1 regularization, dropout, and early stopping [63].

We explore in this work an alternative that constrains the structure of the neural network by, for example, using a neural network with an uneven number of neurons at different layers.

To this end, we propose an *a posteriori* technique that transforms a rectangular network to a non-rectangular one based on the statistical clustering of the output variables at each layer. The intuition is that, for a feedforward network, if the outputs of two neurons are highly correlated, the two neurons function the same statistically. Thus, they can be combined.

Generalizing beyond two neurons, if the outputs of a subset of neurons are highly correlated, these group of neurons may be combined as one. A similar idea of pruning a trained network was explored earlier in [65].

A key component of clustering is to define a similarity measure. To this end, we considered a standard similarity measure $\rho_{X,Y}$ between random variables X and Y defined by

$$\rho_{X,Y} = 1 - \frac{\mathbb{E}(XY)}{\sqrt{\mathbb{E}(X^2)\mathbb{E}(Y^2)}}$$

A variety of clustering techniques can be applied once a similarity measure is chosen. In particular, hierarchical agglomerative clustering [66] allows us to control the number and the sizes of clusters.

In Section 3.2.4, we present numerical results on the structure of neural network using the above clustering analysis. The results from these experiments suggest that a network with bulging middle section seems to perform somewhat better.

3.2.2.2 Learning Net Injection Distributions

The distribution of net injection needs to be learned in order to generate training samples. Distribution learning can be parametric or non-parametric [67],[68]. By restricting the distribution class, parametric techniques are well developed. The assumption on the parametric class can be wrong, however, in which case very little can be said about the performance. This makes it highly desirable that the Bayesian estimator that relies on distribution information is robust against distribution mismatch.

Because of the historical data for injections are limited, we estimate the injection distribution based on the parametric model of Gaussian mixtures that is commonly used to model load and renewable generations [16][69]. The maximum likelihood (EM) method is used in the estimation.

If power injection can be measured locally at injection points, the injection distribution can be estimated locally without communicating the data to the central location; only distribution parameters need to be transmitted. However, if measurements of net injection are not available, we propose an alternative technique that uses smart meter data to estimate the power injection distributions. This is nontrivial because smart meters typically measure accumulative consumptions, and smart meter data are collected at a much slower time scale. In the appendix, we present a time-series based technique that exploits the underlying structure of the Gaussian mixture distributions. This technique converts the Gaussian mixture parameters of the smart meter data distributions to distribution parameters for the fast time-scale power injections.

3.2.3 Bad Data Detection and Data Cleansing

State estimation relies on data, and the quality of data depends on the type of metering, the quality of the collection and communication systems, and the possibility of cyber data attacks where measurement data are manipulated by an attacker [45]. Therefore, bad data detection and cleansing are an integral part of power system state estimation.

The Bayesian formulation offers a direct way to detect, identify, and clean bad data. In contrast to the conventional bad data detection methods, Bayesian bad data detection and cleansing occur *prior to* state estimation by exploiting the learned (prior) probability distributions of the measurements. Under the Bayesian setting, bad data detection problem is one of testing whether a given data sample comes from (or consistent with) the prior distributions of the measurements.

We formulate a binary hypothesis testing problem where hypothesis $\mathcal{H}_0$ models measurements without bad data and $\mathcal{H}_1$ for measurements with bad data. Consider first the simpler case when, under $\mathcal{H}_0$, measurement Y is Gaussian with mean μ_0 and variance σ_0^2 . Under $\mathcal{H}_1$, Y has a different distribution with mean or variance unequal to those under $\mathcal{H}_0$. This is a composite hypothesis testing problem for which the uniformly most powerful test may not exist.

A widely used practical scheme is the Wald test [67] based on testing the normalized deviation of the measurement away from μ_0 ; $\mathcal{H}_1$ (bad data) is declared when the deviation exceeds a certain threshold. Specifically, given $Y = y$, the size α Wald test is given by

$$\left| \frac{y - \mu_0}{\sigma_0} \right| \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\geq}} z_{\alpha/2} := Q^{-1}(\alpha/2)$$

where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-u^2/2) du$ is the Gaussian tail probability function. Typically, the size parameter α is set at $\alpha = 0.05$ to ensure that the false alarm (false positive) probability is no greater than 5%.

For the application at hand, the mean μ_0 and variance σ_0^2 of the measurement under $\mathcal{H}_0$ (no bad data) used in the Wald test can be easily learned as a by-product of the neural network training process described in Section 3.2.2. Specifically, the solutions of the power flow equations give directly samples of the measurements, from which the mean and variance can be estimated.

Strictly speaking, however, the measurement distributions are not Gaussian, and an alternative to the Wald test can be derived by using explicitly the learned distribution. This appears to be unnecessary from our simulation examples shown in Section 3.2.4.

The problem of Bayesian data cleansing, once a bad data sample is detected, can be easily solved by replacing the bad data measurement with μ_0 — the mean of the prior distribution — that has already been learned in the training process.

3.2.4 Simulations Results and Discussions

3.2.4.1 Simulation Settings

The System Simulated

The simulations were performed on an 85 bus low voltage system defined in the MATPOWER toolbox [70]³. To model the relatively high penetration of solar, 55 of the buses were chosen to have solar PV attached to the load; 82 of the buses were assumed to have consumptions.

Three types of measurement devices were assumed: (i) Smart meters were used to measure the energy consumption of every load bus; they produced the average (or total) net injection every 15 minutes. (ii) Current magnitude meters were placed in 20% of distribution system branches to measure the current magnitudes; each meter produced one measurement every second. (iii) One SCADA meter was placed at the slack bus to measure complex power injection to the transmission grid. The SCADA meter produced one measurement per second.

The additive measurement noise was assumed to be independent and identically distributed Gaussian with zero mean and variance set at 1% of the average net consumption value.

Performance Measure

The performance of the tested algorithms was measured by the per-node average squared error (ASE) defined by

$$ASE = \frac{1}{MN} \sum_k \|\hat{x}[k] - x[k]\|^2, \quad (3.5)$$

where M is number of Monte Carlo runs, k the index of the Monte Carlo run, N the number of nodes, $\hat{x}[k]$ and $x[k]$ the estimated and the state vectors, respectively.

Neural Network Specification and Training

The inputs of the neural network were current magnitudes and slack bus measurements; the outputs were the state estimates. The ReLU (Rectified Linear Units) activation function was used for neurons in the hidden layers and linear activation functions in the output layer.

To train the neural network, the Adam algorithm was used with mini batches of 32 samples. To avoid over-fitting, early stopping was applied by monitoring validation errors. To select an initial point for the optimization, He's normal [71] method is used. The training algorithm was implemented in Keras with Tensorflow as the backend [72].

3.2.4.2 Distribution Learning

We used data sets from the Pecan Street collection⁴ in distribution learning. Specifically, a data set covering May 21st to September 21st of 2015 was used for training, and a data set covering the same period of 2016 was used for testing. The data sets were comprised of power measurements

³ The 85 bus system is a single phase system in MATPOWER.

⁴ <http://www.pecanstreet.org/>

from 82 households, from which 55 had solar PV installed. Power consumptions and solar PV generations were measured separately.

Distributions of active power consumptions and solar generations were learned from data separately. We considered non-parametric (histogram and Kernel) and parametric distribution estimation techniques. The latter includes Gaussian, Gaussian mixture, and Weibull models. Among these choices, the Gaussian mixture model with three components was chosen for reasons described earlier. In our simulation, we observed that Gaussian and Weibull distributions may exceed the 95% confidence interval that captures the true distribution.

Figure 3.2 shows a representation of the cumulative distribution estimates of consumption and solar generation at 3 p.m. on bus 12 based on the histogram, 3-component Gaussian mixture, Kernel Epanechnikov estimators, and the 95% confidence bounds. These estimates produce similar results for all distributions and are well within the non-parametric confidence bounds obtained using the Dvoretzky-Kiefer-Wolfowitz inequality [67]. For all cases, 100% of the 3-component Gaussian mixture estimates are within the 96% level confidence bound, and 99.8% of all estimates are within the 85% level confidence bound.

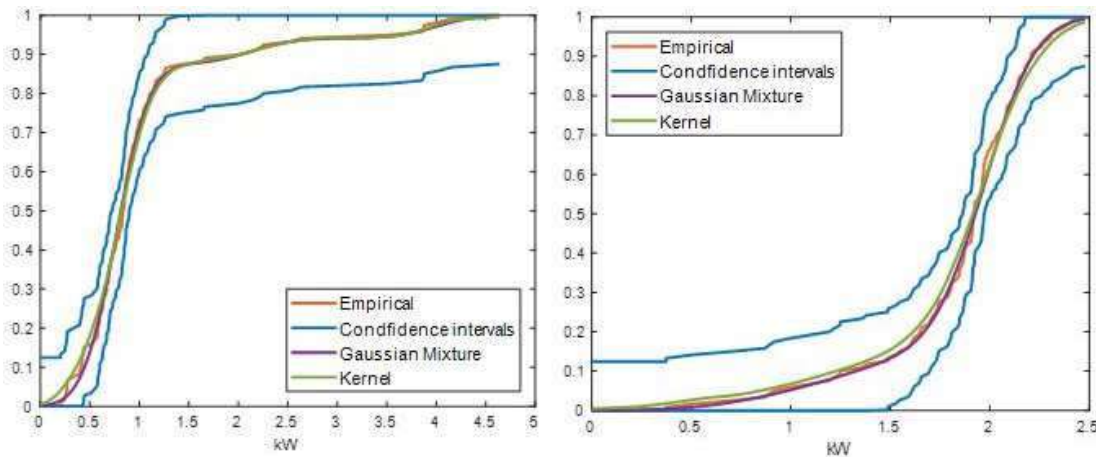


Figure 3.2 Estimated cumulative distribution and 95% confidence interval.

3.2.4.3 Simulation Results

ASE Performance

We compared the proposed *Bayesian state estimation with deep neural network* (herein abbreviated as BSEdnn) with two WLS-based pseudo-measurement methods in the literature:

1. WLS_p: a technique, referred to as *WLS with pseudo-measurements*, that generates injection pseudo-measurement by averaging the energy consumption measurement over a number of samples [13];
2. WLS_{nn}: a method, referred to as *WLS with neural-network pseudo-measurements*, that uses a neural network that generates pseudo-measurements of net injection based on a regression on the last energy consumption vectors [14].

Separate neural networks were implemented and trained for each hour of the day. We generated 10,000 training and 1,000 validation samples. The training of the networks took on average 500 update iterations.

Figure 3.3 (left) shows that the ASE performance of the three state estimators for the 24 hour period. Across all hours, BSEdnn performed significantly better than the two pseudo-measurement-based WLS techniques. Specifically, the ASE of BSEdnn achieved one to two orders of magnitude lower ASE than the pseudo-measurement techniques. The performance gain was attributed to that, although measurement distributions were in some way encoded in the pseudo-measurements, WLS used this information incorrectly. The benefits of leveraging existing WLS software did not appear to justify the loss of performance.

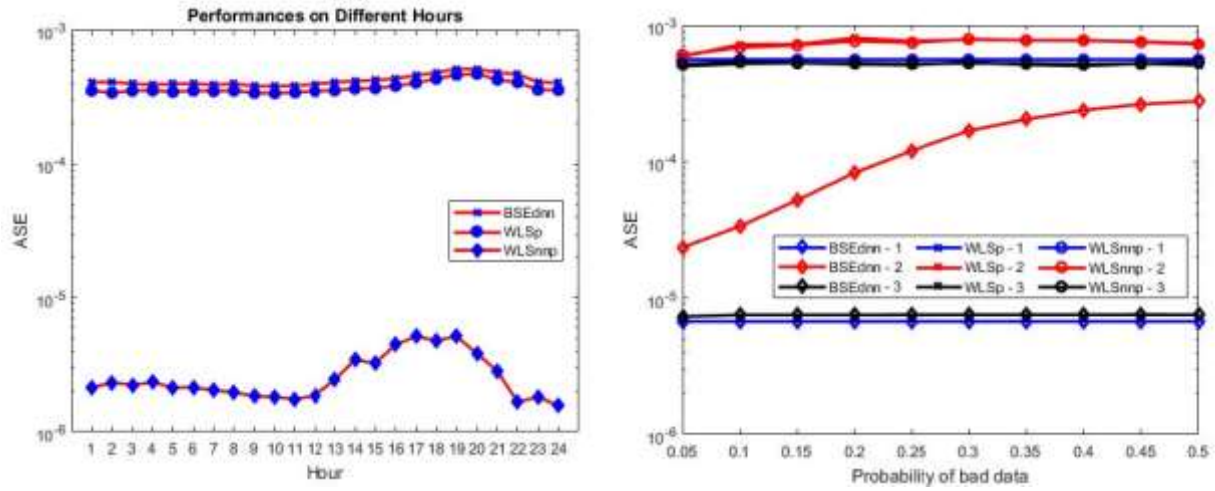


Figure 3.3 ASE performance for three state estimators. Left: ASE (in absence of bad data) of estimators in different hours. Right: ASE at hour 17 in the presence of bad data of different strength; Line 1 shows ASE without bad data, Line 2 shows ASE with bad data and Line 3 shows ASE with bad data and cleansing for estimators.

Bad data detection and cleansing

The performance of bad data detection and cleansing was tested. In the simulations, we generated 1,000 test cases. For each case, bad data were injected to measurements probabilistically; each measurement had probability η to contain bad data. Different strengths of the bad data were evaluated by varying η . Note that the way bad data were introduced deviated from and significantly more challenging than the conventional settings where a fixed number of bad data at specific locations was typically assumed.

For the bad data model considered in Section 3.2.3, we assumed that, under hypothesis $\mathcal{H}_1$ (bad data), the additive noise was Gaussian with zero mean and considerably larger standard deviation σ_1 than σ_0 under $\mathcal{H}_0$ (no bad data)⁵. Once bad data were detected, BSEdnn performed a data

⁵ Note that neither the mean and the standard deviation nor the distribution of the bad data were assumed known in the bad data detection and cleansing algorithm.

cleansing procedure that replaced the bad data with the mean of the measurements as described in Section 3.2.3. For WLS-p and WLS-nnp, once bad data were detected, they were removed from the WLS procedure.

Figure 3.3 (right) shows the ASE performance vs. the probability η of bad data occurrence under the bad data model $\sigma_1 = 10\sigma_0$. For all algorithms, bad data detection and cleansing algorithm were compared with two cases: Case 1 shown in dotted (blue) lines was the baseline performance when there were no bad data; Case 2 shown in dashed (red) lines was the performance when bad data were present but not cleansed. It was evident that bad data detection and cleansing improved the performance of state estimation considerably, driving the ASE performances, shown in solid (black), closer to those for the clean data case. It should also be observed that, even without bad data cleansing, BSEdnn performed better than pseudo-measurement based WLS estimators.

Additional simulations were conducted by varying the strength of the bad data and examining the performance of bad data detection. Recall that, in deriving the threshold used in the Wald test for bad data detection in Section 3.2.3, a Gaussian distribution of the measurement under $\mathcal{H}_0$ was assumed, which at best could only be an approximation given the Gaussian mixture nature of the net injection. To evaluate the performance of bad detection derived from the Gaussian model, we provided a comparison with the case when noise distributions were indeed Gaussian, for which the theoretical value of false alarm and miss detection probabilities could be computed analytically.

Table 3.1 shows bad data detection and cleansing performance. The top two blocks of the table show that the threshold obtained based on the Gaussian model approximated well the false alarm and detection probabilities under the Gaussian mixture models. Also shown here is that the pseudo-measurement schemes with WLS did not perform well. The third and fourth blocks show the effects of bad data on the ASE of state estimators. The proposed BSEdnn appeared to be more sensitive to bad data than the WLS-based methods while the bad data cleansing algorithm mitigated effects of bad data satisfactorily.

Table 3.1 Bad data detection and cleansing with varying strength of bad data. $\eta = 0.3$

	$\sigma_1 = 5\sigma_0$	$\sigma_1 = 10\sigma_0$	$\sigma_1 = 20\sigma_0$
False alarm prob. (Gaussian theoretical)	5.00%	5.00%	5.00%
False alarm rate of Wald test	4.60%	4.70%	4.70%
False alarm rate of WLSp+J(x)	4.94%	4.78%	4.91%
False alarm rate of WLSnnp+J(x)	4.94%	4.78%	4.91%
Detection prob. (Gaussian Theoretical)	69.5%	84.5%	92.1%
Detection rate of Wald test	68.14%	84.16%	91.73%
Detection rate of WLSp+J(x)	36.45%	41.75%	57.33%

Detection rate of WLSnp+J(x)	36.75%	43.37%	58.96%
ASE of BSEdnn without bad data	6.50E-06	6.50E-06	6.50E-06
ASE of WLSp without bad data	5.87E-04	5.87E-04	5.87E-04
ASE of WLSnp MSE without bad data	5.85E-04	5.85E-04	5.85E-04
ASE of BSEdnn with bad data	1.84E-05	9.01E-05	7.31E-04
ASE of WLSp with bad data	6.97E-04	7.06E-04	7.75E-04
ASE of WLSnp with bad data	6.98E-04	6.90E-04	7.70E-03
ASE of BSEdnn with bad data cleansing	6.91E-06	6.91E-06	6.86E-06
ASE of WLSp with bad data cleansing	5.82E-04	5.89E-04	5.79E-04
ASE of WLSnp MSE with bad data cleansing	5.76E-04	5.82E-04	5.79E-04

3.2.4.4 Optimizing the Structure of Deep Neural Network

We examined the effects of choosing different structures of the neural network on the performance of BSEdnn. To this end, we considered the two questions: (i) given a fixed number of neurons, is there an optimal depth of the neural network that gives the best performance? (ii) is there a particular "shape" of the network that should be favored? We attempted to address these issues through simulations, knowing that interpretations drawn here may not be conclusive.

The depth of deep learning

For total 1000, 2000, and 3000 neurons, we tested exhaustively different rectangular neural networks of different depths until a trend was observed. The ASE performance of BSEdnn using validation data sets is shown in Figure 3.4. The results suggested that a neural network of 4 to 6 layers perform the best.

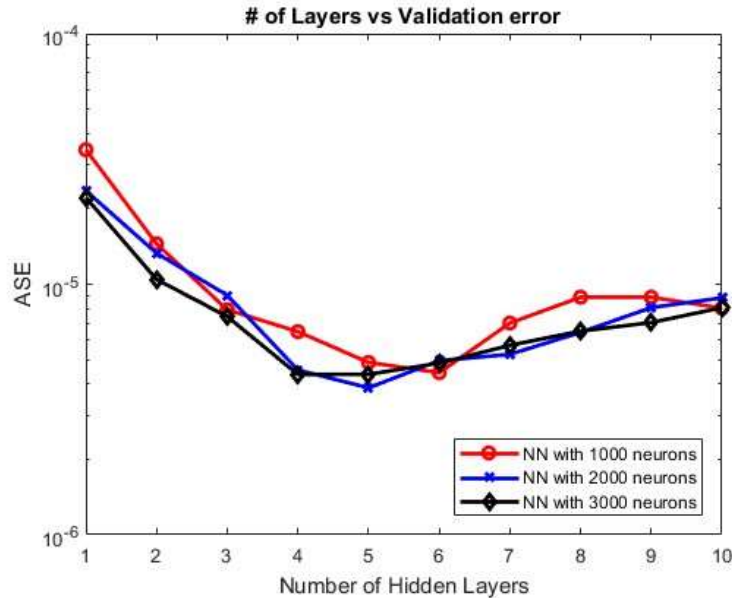


Figure 3.4 NN Architecture - Total number of neurons fixed.

The shape of deep neural network

Next, we examined whether using an irregular shaped neural network has any advantage. To this end, we considered a pruning approach that, starting from a rectangular network, progressively combining neurons that were highly correlated based on the clustering technique developed in Section 3.2.2. The idea was that, by combining neurons whose outputs were highly correlated, the resulting irregular neural network would be regularized within a certain structure that, potentially, could lead to faster convergence and better performance.

Table 3.2 summarizes the simulation results. First, we evaluated the rectangular structure with 400 neurons in each layer. Second, we used the clustering algorithm to prune the network, which resulted in a reduction of the number of neurons at the two ends of the network and improved test error. The second round of clustering-pruning resulted in further reduction training, test, and validation errors, and a shape of the network with the second and third layers having considerably more neurons than the first and last two layers. The third round of clustering-pruning resulted in an increase of errors, indicating the locally best clustering-pruning procedure should end at the second iteration.

Table 3.2 Pruning simulation results with the threshold = 0.0005

	Rectangular	1st Pruning	2nd Pruning	3rd Pruning
# of Neurons 1st layer	400	399	395	391
# of Neurons 2nd layer	400	400	400	400
# of Neurons 3rd layer	400	400	399	380
# of Neurons 4th layer	400	346	278	173
# of Neurons 5th layer	400	154	64	22
Test error	6.23e-06	5.77e-06	4.45e-06	5.11e-06
Validation error	4.73e-06	4.64e-06	3.72e-06	3.85e-06
Training error	4.27e-06	4.13e-06	3.41e-06	3.73e-06

3.2.5 Conclusion

This chapter presents a Bayesian approach to state estimation and bad data cleansing using deep neural networks in unobservable distribution systems. The proposed approach employs two machine learning techniques: distribution learning of power injection and deep learning of MMSE estimator by embedding power flow equations. A Bayesian bad data detection and cleansing algorithm has also been developed. Simulation results demonstrate the potential of the Bayesian state estimation for cases that are intractable for conventional WLS-based techniques. The use of deep neural network plays a key role in overcoming the computation complexity in Bayesian estimation. Once learned, the neural network MMSE estimator has a linear complexity solution for state estimation, which makes it suitable for real-time applications.

The Bayesian approach presented here has its limitations; research and evaluations outside the scope of this report are needed. First, because prior distribution plays such an important role in Bayesian techniques, the estimator is less capable of adapting to changes in the network such as

line and generation outages. Second, the training of deep neural network is an area of research undergoing intensive investigation. Yet, techniques available are largely ad hoc with little guarantee of performance. A large number of parameters have to be tuned. For this reason, while the Bayesian principle on which this work is based is sound and promising, the performance reported here is a result of try-and-error tuning of design parameters.

3.2.6 Appendix

We present here a technique that estimates the power injection distribution using data collected from smart meters. The main technical challenge is that smart meters measure the *cumulative energy injections (kWh)* at a time scale slower than that of the SCADA measurements.

For convenience, assume that the fast time-scale data (SCADA) are produced every interval of unit duration whereas the smart meter measurements are produced once every T intervals. Let the sequence of the smart meter measurements be Y_t . The t th interval has T fast-time measurements $X_{t1}, \dots, X_{tT}$. We have

$$Y_t = \sum_{n=1}^T X_{tn}. \quad (3.6)$$

Here we make the assumption that (X_{ti}) is a stationary process which makes Y_t also stationary.

Empirical studies have suggested that a good model for the distribution f_Y of Y_t is a Gaussian mixture, *i.e.*,

$$f_Y = \sum_{i=1}^K \pi_i \mathcal{N}(m_i, V_i^2),$$

where $\mathcal{N}(m, V^2)$ denotes the Gaussian distribution with mean m and covariance V^2 . Such a model has the interpretation that there are K weather patterns, each can be modeled by a Gaussian process. With probability α_i , the i th pattern shows up in the interval of measurement. Under this interpretation, it is reasonable to assume that X_{ti} also follows the same underlying physical model: the distribution f_X of X_{ti} is also a Gaussian mixture with the same number of components and the same probability of appearance, *i.e.*, $f_X = \sum_{i=1}^K \pi_i \mathcal{N}(\mu_i, \sigma_i^2)$. Therefore, we only need to find μ_i from m_i and σ_i^2 from V_i^2 .

We now fix t and a particular Gaussian mode i to derive a procedure to compute σ_i^2 from V_i^2 . To avoid cumbersome notations, we drop the component index i in the Gaussian mixture and the subscript t , writing X_{tn} as X_n , V_i^2 as V^2 , and σ_i^2 as σ^2 .

From (2.6), we have immediately $\mu_i = \frac{1}{T} m_i$. Let $C_x(k)$ be the auto-covariance of X_n with lag k . Then, the variance V^2 of Y_t satisfies

$$V^2 = \sum_{i,j=-T+1}^{T-1} C_x(i-j) = \gamma^T c, \quad (3.7)$$

Where

$$c := [C_x(0), \dots, C_x(T-1)]^T, \\ \gamma := [T, 2(T-1), 2(T-2), \dots, 2]^T.$$

Our goal is to obtain $\sigma^2 = C_x(0)$ from V^2 .

We now make the assumption that X_n is a stationary AR- K process defined by

$$X_n = \alpha_1 X_{n-1} + \dots + \alpha_K X_{n-K} + \varepsilon_n, \quad (3.8)$$

where $\varepsilon_n \in \mathcal{N}(\mu_\varepsilon, \sigma_\varepsilon^2)$ is the innovation sequence that is IID Gaussian. With a one or more traces of X_n , the AR parameters $\alpha = (\alpha_k)$ can be easily estimated using, *e.g.*, the least squares method.

Knowing α , the auto-covariance functions can be computed directly from a variation of the Yule-Walker equation:

$$c = A_\alpha c + \sigma_\varepsilon^2 e_1 \Rightarrow c = \sigma_\varepsilon^2 (I - A_\alpha)^{-1} e_1,$$

where $e_1 = [1, 0, \dots, 0]^T$, and A_α is a matrix made of entries of α . Substituting c into (3.7), we obtain

$$\sigma^2 = \frac{e_1^T (I - A_\alpha)^{-1} e_1}{\gamma^T (I - A_\alpha)^{-1} e_1} V^2.$$

To summarize, from the smart meter data Y_t , we obtain first the Gaussian mixture coefficients $\{(\pi_i, m_i, V_i^2)\}$, which gives the coefficients $\{(\pi, \mu_i, \sigma_i^2)\}$ of the Gaussian mixture of X_{tn} . Specifically,

$$\mu_i = \frac{1}{T} m_i, \quad \sigma_i^2 = \frac{e_1^T (I - A_\alpha^{(i)})^{-1} e_1}{\gamma^T (I - A_\alpha^{(i)})^{-1} e_1} V_i^2 \quad (3.9)$$

where $\alpha^{(i)}$ is the coefficients of the AR process associated with component i .

3.3 Anomaly Data Sequence Detection via ICA-GAN

3.3.1 System and Anomaly Detection Model

3.3.1.1 Measurement and Anomaly Models

Let the measurement sequence⁶ at sensor i be (Z_{it}) , which we model as a random process generated from power system state sequence (x_t) , additive noise (w_{it}) , and anomaly sequence (a_{it}) :

⁶ We adopt the standard notation that (x_t) denotes a sequence of measurements.

$$z_{it} = h_i(x_t) + w_{it} + a_{it}, \quad (3.10)$$

where the measurement function $h_i(\cdot)$ at sensor i encodes system parameters and topology information. Herein, we make the assumption that noise processes (w_{it}) at different sensors are statistically independent whereas the anomaly sequences (a_{it}) may be dependent.

For natural data anomalies, (a_{it}) are assumed to be independent of ($h_i(x_{it})$), ambient noise (w_{it}) and measurements elsewhere (z_{jt}). For adversarial data anomalies, very little can be assumed about a_{it} . In particular, a_{it} may be a function of past measurements and statistically dependent on the system state in some arbitrary fashion. An extreme type of unobservable attack can be constructed in the form of $a_{it} = h_i(x'_t) - h_i(x_t)$ where the adversary substitutes the actual system measurement $h_i(x_t)$ by a fictitious measurement corresponding to a fictitious state x' . For system anomalies with post-contingency measurement function $h'_i(\cdot)$, the anomaly sequence can simply be $a_{it} = h'_i(x_t) - h_i(x_t)$.

3.3.1.2 Sensor Level Anomaly Sequence Detection

At the sensor level, we formulate the anomaly sequence detection problem as a non-parametric hypothesis testing of a time series. We assume that at time t , we have a block of M of current and past measurements $Z_{it} = (z_{it}, z_{i(t-1)}, z_{i(t-M+1)})$. Let the null hypothesis $\mathcal{H}_0$ model the anomaly-free data and the alternative $\mathcal{H}_1$ for the anomaly data. In particular

$$\begin{aligned} \mathcal{H}_{i0} : Z_{it} \sim f_{i0} \text{ vs. } \mathcal{H}_{i1} : Z_{it} \sim f_{i1} \in \mathcal{F}_{i,\varepsilon} \\ \mathcal{F}_{i,\varepsilon} := \{f, \|f - f_{i0}\| \geq \varepsilon\} \end{aligned} \quad (3.11)$$

where f_{i0} and f_{ii} are the underlying joint probability distributions of Z_{it} under $\mathcal{H}_{i0}$ and $\mathcal{H}_{i1}$, respectively. Note that $\mathcal{H}_{i0}$ is a simple hypothesis with a single probability distribution f_{i0} and $\mathcal{H}_{i1}$ a *composite hypothesis* with a set $\mathcal{F}_\varepsilon$ of distributions some ε distant away⁷. The requirement of ε separation of the null and the alternative hypothesis is to ensure consistency of the detector.

Under (3.11), each sensor makes an individual binary decision $u_{it} = \mathcal{D}_i(Z_{it}) \in \{0,1\}$ on anomaly based on Z_{it} : $u_{it} = 1$ means that the detector rejects the null (anomaly-free) hypothesis $\mathcal{H}_{i0}$ and $u_{it}=0$ means that the null hypothesis $\mathcal{H}_{i0}$ is accepted. In practice, a sensor produces a detection every M samples when non over-lapping blocks are used. The size of M has both theoretical and practical implications. A larger M means better detection reliability with considerably higher complexity in learning and implementation.

⁷ The distance measure of probability distributions can be arbitrary. Examples include the total variation and Jensen-Shannon distances.

3.3.1.3 Distributed Detection and Data Fusion Model

We now consider a power system with K local PMU/CPOW sensors as shown in Figure 3.5, where sensor i produces a local binary sensor detection U_{it} at time t . We assume that local decisions $\{u_{it}\}$ are communicated synchronously to the control center (fusion center) where a global decision v_t on anomaly is made. Let $u = (U_{1t}, \dots, U_{Kt})$ be the local decision vector at the control center.

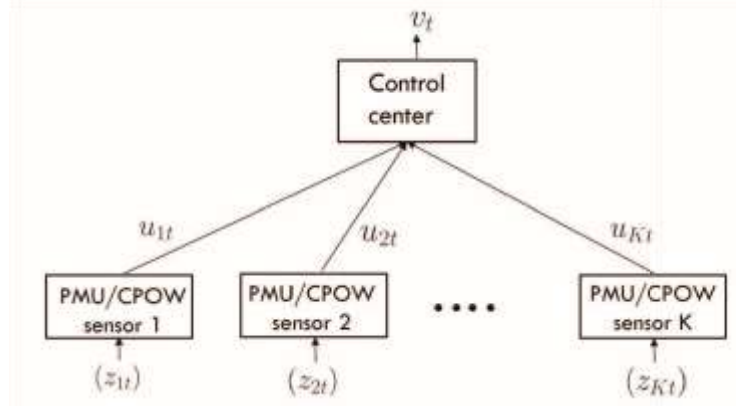


Figure 3.5 A schematic of a distributed anomaly detection system.

Assume that the local detector at sensor i has false positive rate (FPR) of α_i and true positive rates of β_i , the control center faces the following binary hypothesis testing problem $\mathcal{H}_0$ vs. $\mathcal{H}_1$ where $\mathcal{H}_0$ corresponds to the anomaly-free hypothesis and $\mathcal{H}_1$ the anomaly:

$$\mathcal{H}_0 : u_t \sim P_0 \text{ vs. } \mathcal{H}_1 : u_t \sim P_1, \quad (3.12)$$

where $u_{it} \sim B(\alpha_i)$ is Bernoulli⁸ with parameter α_i under $\mathcal{H}_0$ and $u_{it} \sim B(\beta_i)$ under $\mathcal{H}_1$. In (3.12), P_0, P_1 are joint probability mass functions of u_t under $\mathcal{H}_0$ and $\mathcal{H}_1$, respectively. Note that, while the FPR α_i of is a design parameter that can be controlled, the TPR β_i is unknown, and it varies with the realized anomaly.

3.3.2 Sensor-Level Anomaly Sequence Detection

We now focus on the anomaly detection problem at a particular sensor. For brevity, we drop the sensor index i in the subscripts of relevant variables.

Figure 3.6 shows a schematic of the proposed technique, which includes an independent component analysis (ICA) preprocessing $\mathcal{G}_\theta$ and a uniformity test. At time t , a vector consisting of M measurements $Z_t = (z_t, z_{t-1}, \dots, z_{t-(M-1)})$ is passed through a neural network trained to extract a block of uniformly distributed independent components $V_t = (v_{t,1}, \dots, v_{t,N})$ under the anomaly-free model (3.11). The training of ICA-GAN is discussed in Section 3.3.2.2, where either an offline or online training using past anomaly-free measurements can be used.

⁸ The Bernoulli random variable $X \sim B(p)$ is defined here by $\Pr(X = 1) = p$ and $\Pr(X = 0) = 1 - p$.

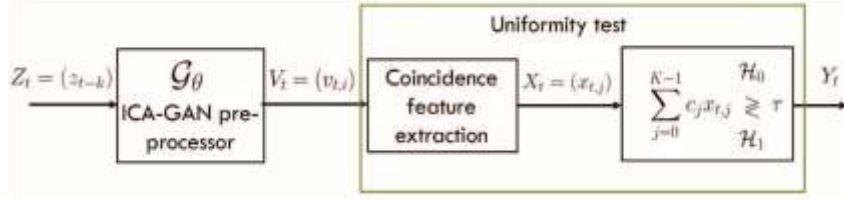


Figure 3.6 A schematics of ICA-GAN for anomaly sequence detection.

The uniformity test takes the output of the ICA preprocessor $\mathcal{G}_\theta$ and produces a K -dimensional coincidence feature $X_t = (x_{t,0}, \dots, x_{t,K-1})$ followed by a linear classifier whose output Y_t labels the input sequence Z_t as anomaly ($Y_t = 1$) or anomaly-free ($Y_t = 0$).

The implementations of the ICA preprocessing and uniformity test are described next.

3.3.2.1 Anomaly Sequence Detection via Uniformity Test

We begin with the uniformity test for anomaly sequence detection, assuming that the preprocessing step has generated V_t that, under the anomaly-free hypothesis $\mathcal{H}_0$, contains *i.i.d.* uniformly distributed random variables $(v_{t,i})$ in $[0,1]$.

To derive the detection feature vector X_t , we first quantize $v_{t,i}$ uniformly into a discrete random variable $\bar{v}_{t,i}$ of alphabet size K , *i.e.*,

$$\bar{v}_{t,i} = k \text{ if } v_{t,i} \in B_k = [k/K, (k+1)/K),$$

where we refer B_k to as the k -th *quantization bin*. Such a quantization transforms the original anomaly detection problem (3.11) to the classical uniformity test defined as

$$\begin{aligned} \mathcal{H}_0' : (\bar{v}_{t,i}) &\stackrel{\text{i.i.d.}}{\sim} P'_0 = \left(\frac{1}{K}, \dots, \frac{1}{K}\right), \\ \mathcal{H}_1' : (\bar{v}_{t,i}) &\sim P'_1 \in \mathcal{F}_{\varepsilon'}, \end{aligned} \quad (3.13)$$

where $F_{\varepsilon'} = \{p = (p_1, \dots, p_K) \mid \|p - P_0\| > \varepsilon\}$. Note that the probability distribution under $\mathcal{H}_0$ above is known whereas f_0 in (3.13) is unknown.

Following the classic work of David [54] and Viktorova and Chistyakov [55], we define a K -dimensional detection feature vector $X_t = (x_{t,0}, \dots, x_{t,K-1})$ where $x_{t,k}$ is the number of quantization bins that have exactly k samples of $(\bar{v}_{t,1}, \dots, \bar{v}_{t,N})$. In particular, $x_{t,0}$ is the number of quantization bins that contains no samples of $(\bar{v}_{t,i})$ and $x_{t,1}$ the number of quantization bins containing one sample.

With the feature vector X_t , a linear anomaly detector for hypothesis testing (3.11) is given by

$$\sum_{k=0}^N c_k x_{t,k} \underset{\mathcal{H}_1'}{\overset{\mathcal{H}_0'}{\geq}} T_\alpha, \quad (3.14)$$

where T_α is the threshold that controls the level of false positive detection rate.

In [56], Paninski shows that the above detector is consistent when only $x_{t,0}$ is used ($c_k = 0, k > 1$) so long as N grows faster than $\sqrt{K}$ as $N = o(\frac{1}{\varepsilon^4} \sqrt{K})$. Remarkably, the sample complexity can be significantly less than the size of the alphabet. When the coefficients of the linear detector is carefully chosen as in [55], the detector in (3.14) is asymptotically most powerful.

The threshold of the test statistics affects the true and false-positive probabilities of the detection. The threshold T_α of the K_1 coincidence test with the constraint on the false-positive probability to no greater than α is given by

$$T_\alpha = \min\{k : \Pr(K_1 \leq k; \mathcal{H}_0) \leq \alpha\}. \quad (3.15)$$

The computation of T_α amounts to evaluating $P_k := \Pr(K_1 = k; \mathcal{H}_0)$, which was given by Von Mises in [73]:

$$P_k = \sum_{j=k}^K (-1)^{j+k} \binom{j}{k} \binom{K}{j} \frac{N!}{(N-j)!} \frac{(K-j)^{N-j}}{K^N}.$$

3.3.2.2 Extracting Independent Components via ICA-GAN

Independent Component Analysis (ICA), a generalization of Principle Component Analysis (PCA), extracts a set of independent components from a block of measurements. Originally proposed by Jutten and Herault [51] and Comon [52], ICA has found a wide range of applications when statistical independence is essential in learning and inference tasks. ICA typically requires nonlinear processing, and neural network techniques have been proposed [74]. More recently, Brackel and Bangio introduced a deep learning solution based on GAN [53]. However, their technique that enforces independence through resampling does not perform well in time series. Here we propose an alternative solution, referred to as ICA-GAN, based on direct minimization of Wasserstein distance [44] between the distribution of ICA estimates and that of uniform *i.i.d.* random variables.

Assume that the measurement vector has a nonlinear ICA representation [75],

$$Z_t = f(\tilde{V}_t), \quad (3.16)$$

where $\tilde{V}_t = (\tilde{v}_{t,1}, \dots, \tilde{v}_{t,N})$ has uniform *i.i.d.* components $\tilde{v}_{t,i} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}(0,1)$. The proposed ICA-GAN produces a minimum Wasserstein-distance estimate V_t of $\tilde{V}_t$.

The learning structure of ICA-GAN, shown in Figure 3.7, is an *inverse GAN*⁹, where the ICA-GAN neural network $\mathcal{G}_\theta$ with weights vector θ , once properly trained, maps a sequence of arbitrary

⁹ The standard GAN trains a generative network that transforms a uniform distribution to an underlying distribution of a data set.

distributed random variables to a uniform *i.i.d* sequence. A discriminator neural network with $\mathcal{D}_\eta$ weights vector η , through a dual optimization, computes the estimated gradient of Wasserstein distance between the distribution of the estimated ICA V_t and that of $\tilde{V}_t$ with uniform *i.i.d*. components. The stochastic gradient of the Wasserstein distance is used to update generator neural network coefficient θ and discriminator neural network coefficients η . See an implementation of ICA-GAN in Algorithm 1.

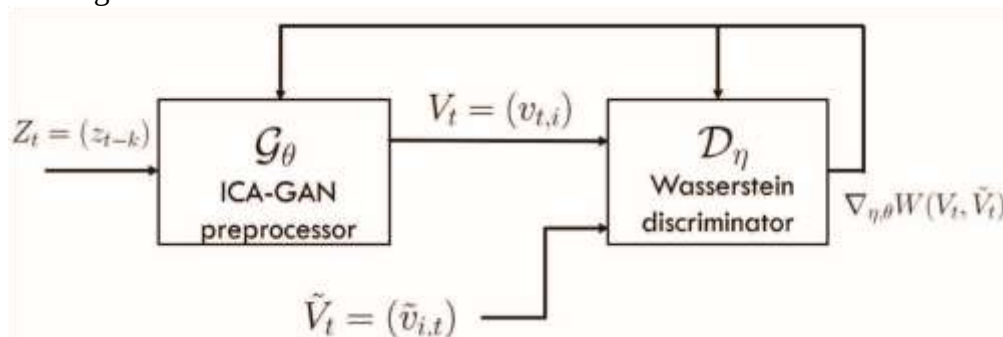


Figure 3.7 Learning structure of ICA-GAN.

Algorithm 1 ICA-GAN. The experiments in this work used the values $\alpha = 0.0001$, $\lambda = 0.1$, $b = 100$, $c = 10$, $M = 80$, $N = 50$.

Require: α , the learning rate. λ , the gradient penalty coefficient. b , the batch size. c , the number of iterations of the discriminator per generator iteration. M , the block size for the data sequence. N , the number of the independent components for ICA.

```

1: for Number of training iterations do
2:   for  $k = 1, \dots, c$  do
3:     for  $i = 1, \dots, b$  do
4:       Sample  $U = (U_1, \dots, U_N) \sim \mathcal{U}^{(N)}(0, 1)$  from uniform distribution.
5:       Sample a random time  $t$  for the start of the time sequence. Get  $Z_t =$ 
          $(z_t, z_{t-1}, \dots, z_{t-(M-1)})$  measurements sequence from data sequence.
6:       Sample a random number  $\varepsilon \sim \mathcal{U}(0, 1)$ .
7:        $\tilde{U} \leftarrow g_\theta(Z_t)$ 
8:        $\hat{U} \leftarrow \varepsilon U + (1 - \varepsilon)\tilde{U}$ 
9:        $L_i \leftarrow f_\omega(\tilde{U}) - f_\omega(U) + \lambda(\|\nabla_{\tilde{U}} f_\omega(\tilde{U})\|_2 - 1)^2$ 
10:    end for
11:    Update the discriminator parameter  $\omega$  by descending its stochastic gradient:
          $\omega \leftarrow Adam(\nabla_\omega [\frac{1}{b} \sum_{i=1}^b L_i])$ 
12:  end for
13:  for  $i = 1, \dots, b$  do
14:    Sample a random time  $t$  for the start of the time sequence. Get  $Z_t =$ 
          $(z_t, z_{t-1}, \dots, z_{t-(M-1)})$  measurements sequence from real data sequence.
15:     $L_i \leftarrow -f_\omega(g_\theta(Z_{\{t, \dots, t+(M-1)\}}))$ 
16:  end for
17:  Update the ICA-GAN generator parameter  $\theta$  by descending its stochastic gradient:
          $\theta \leftarrow Adam(\nabla_\theta [\frac{1}{b} \sum_{i=1}^b L_i])$ 
18: end for

```

Ideally, if an ICA representation of the measurement exists, and the training of ICA-GAN preprocessor converges, g_θ transforms the unknown measurement distribution under $\mathcal{H}_{i_0}$ in (3.11) to the uniform *i.i.d.* distribution in V_t . In practice, however, ICA-GAN is trained with historical data samples from the anomaly-free model. With a sufficiently high-dimensional implementation and adequate training, we expect approximately uniform *i.i.d.* entries of vector V_t .

3.3.3 System-Level Anomaly Detection

We now consider the distributed detection problem at the system level where the control center receives binary decisions $\{u_{it}\}$ from individual sensors. From Section 3.3.3, we know that ICA-GAN at each sensor transforms different sensor measurements Z_{it} to the same uniform *i.i.d.* samples under $\mathcal{H}_0$. Because the uniformity detector at all sensors are identical, they all have same false positive rate $\alpha = \alpha_i$. Furthermore, because noise process (w_{it}) are statistically independent

across sensors, we have, under $\mathcal{H}'_0$, u_{it} are *i.i.d.* Bernoulli $B(\alpha)$, and $u_t = \sum_i u_{it}$ a binomial random variable $Bin(K, \alpha)$.

We derive next a Neyman-Pearson detection rule at the control center given detection vector $u_t = (u_{1t}, \dots, u_{Kt})$ under the standard conditional independent assumption, *i.e.*, conditional on $\mathcal{H}'_k$, u_t have independent entries. Because anomaly model is arbitrary, we further assume that the true positive rates β_i for all detectors are the same. Let $\beta = \beta_i$.

$$\begin{aligned}\mathcal{L}(u_t) &= \log \frac{\Pr(u_t | \mathcal{H}'_1)}{\Pr(u_t | \mathcal{H}'_0)} \\ &= \log \frac{\beta(1-\alpha)}{\alpha(1-\beta)} \sum_i u_{it} + K \log \frac{1-\beta}{1-\alpha}.\end{aligned}$$

Noting that $\beta > \alpha$ for any reasonable local detector, the Neyman-Pearson test is given by a threshold on the sum of sensor decision variables $\sum_i u_{it}$:

$$\sum_{i=1}^K u_{it} \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\geq}} \tau, \quad (3.17)$$

where τ is chosen to satisfy the false positive rate constraint. Given the desired upper bound α_0 on the false positive rate of the central detector, we set

$$\tau = \min \left\{ k : \alpha_0 \leq \sum_{j=k}^K \binom{K}{j} \alpha^j (1-\alpha)^{(K-j)} \right\},$$

where we ignore possible randomizations to make the false positive exactly α_0 .

Note that the detector defined in (3.17) is uniformly most powerful (UMP) under the assumptions that sensor-level detectors produce (conditionally) independent decisions with identical TPR. Note also that, although we assume that sensors synchronously communicate their local decisions $\{u_{it}\}$, the above derivation shows that the central detector can just as well operate *asynchronously*. The structure of the detector and the decision rule (3.17) remain the same. Indeed, the above idea also applies to local sensor decisions where the sensor combines multiple detections from smaller blocks to produce more reliable detections. The advantage is that training ICA-GAN with a low-dimensional input vector is considerably simpler than training a higher dimensional one.

3.3.4 Numerical Case Studies

We present three case studies that cover the three types of anomalies considered in this work. Wherever possible, publicly available real data sets were used.

ICA-GAN implementations in the three case studies shared the same structure, although parameters used are tuned differently depending on the training data. The specific data sets used

in the case studies are described in their respective subsections. In all three case studies, we trained the generator with three hidden layers and 100 neurons at each hidden layer. Hyperbolic tangent in Case I and Rectified Linear Units (ReLU) in Case II-III activation function at the final layer were used as the activation functions. For the discriminative network, we also used three hidden layers with 100 neurons. A modification of a standard implementation of Wasserstein distance¹⁰ was used in the ICA-GAN training with Adam optimization [64] algorithm using mini-batches of 100 data samples.

In performance evaluation, we obtained the *receiver operating characteristic* (ROC) curves over Monte Carlo simulations. ROC curves plot TPR (probability of detection) against FPR (probability of false alarm), which shows the detection power across the entire range of FPR constraints. We paid specific attention to FPR=0.05 as in standard power system applications [10].

3.3.4.1 Benchmark Techniques and Implementations

While there are few comparable techniques in the literature for detecting general sequence anomalies in CPOW and high-resolution PMU measurements, we compared three benchmarks that have similar characteristics with ICA-GAN and are potentially applicable in the applications considered in the case studies presented here.

The normalized residue test (NRT) [10] is the classic technique for bad-data detection for power system state estimation. NRT collects measurements from the local sensors and form a centralized anomaly detection. When multiple anomalies occur simultaneously, a standard approach is to remove bad data recursively. In our implementation, we apply the NRT-test to isolate the measurement with the largest total-residue-error calculated over the sequence. If it failed the NRT-test, the data would be declared bad and removed from the system until either the measurement data pass the test or the system becomes unobservable.

The one-class support vector machine (OC-SVM) [42] is a semi-supervised machine learning method trained with anomaly-free historical data. It operates under a similar set of model assumptions, except that it does not deal with temporal dependencies in data. We used the radial basis function as a nonlinear kernel. We evaluated the results on the test sequences using the anomaly score function we achieved. We varied the threshold parameter of SVM to get different points on the ROC curve. We used the scikit-learn library for the implementation.

The fast unsupervised anomaly detection (F-AnoGAN) [43] is an auto-encoder technique trained on anomaly-free data. We used a generator and a discriminator with three hidden layers in Wasserstein GAN and a deep neural network with two hidden layers and 100 neurons in each layer in the auto-encoder. The input took 80 consecutive measurement samples and encoded them into latent variables of dimension to 50. We evaluated the results considering the reconstruction error of the auto-encoder. The training was done on a GPU using the Tensorflow-GPU library [76].

¹⁰ https://keras.io/examples/generative/wgan_gp/

3.3.4.2 Case I: System Anomalies in CPOW Measurements

We used the EPFL data set involving a battery energy storage system connected at a bus [59], as shown in Figure 3.8 (top left). The battery system produced injections that emulated different levels of anomaly events. We used CPOW measurements on the bus voltage and current measurements A and B. Direct measurements on anomaly current at C were not used. The CPOW measurements were direct samples of the voltage/current waveforms at 50kHz, and the anomaly power injection varied from 0 to 500kW. The EPFL data set contained anomaly and anomaly-free data, each with 100,000 samples within 2 seconds.

Figure 3.8 (top-right and the bottom panel) shows the anomaly and anomaly-free waveforms of the bus voltage and current measurements. There is little difference between the anomaly and anomaly-free voltage CPOW measurements, while noticeable differences are shown in the current measurements. It was expected that the two current detectors would be more reliable than the voltage detector. However, the control center would not know which detector would be reliable a priori.

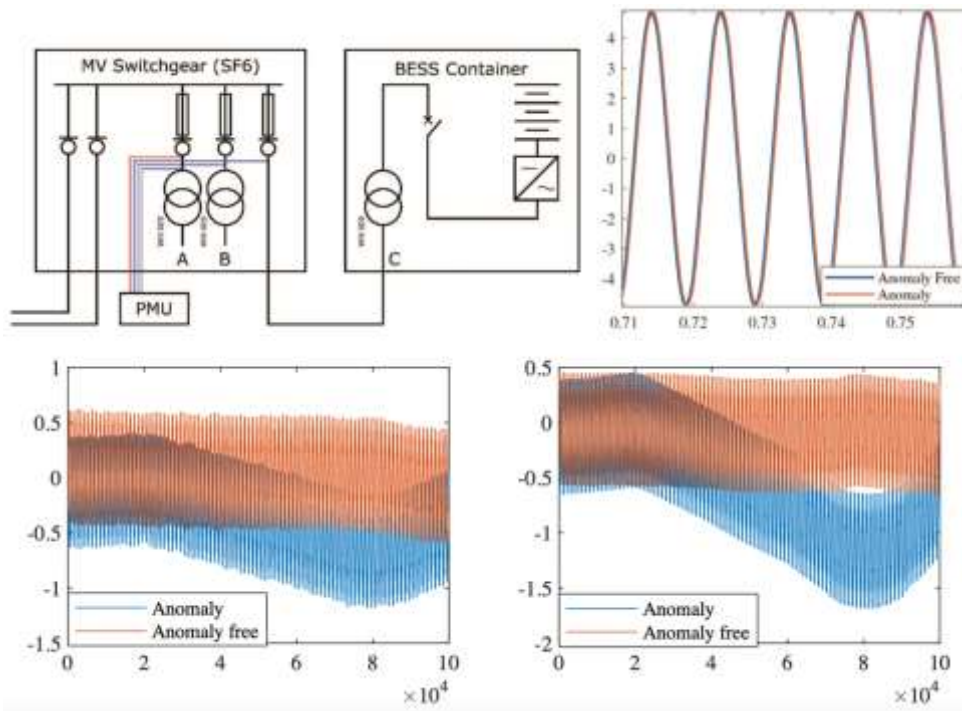


Figure 3.8 Learning structure of ICA-GAN.

Three sensor-level detectors were implemented using the bus voltage and current measurements at A and B. The anomaly-free training data were separated into training and testing sets of the ratio 6:4. The training set contained 120 batches of 500 consecutive samples and the test set 80 batches. For each batch of samples, ICA-GAN generated a batch of preprocessed samples on which uniformity tests were made. A single decision was made by each sensor every 0.01s. The thresholds for the sensor-level detection were chosen such that their FPRs were all equal to 0.2.

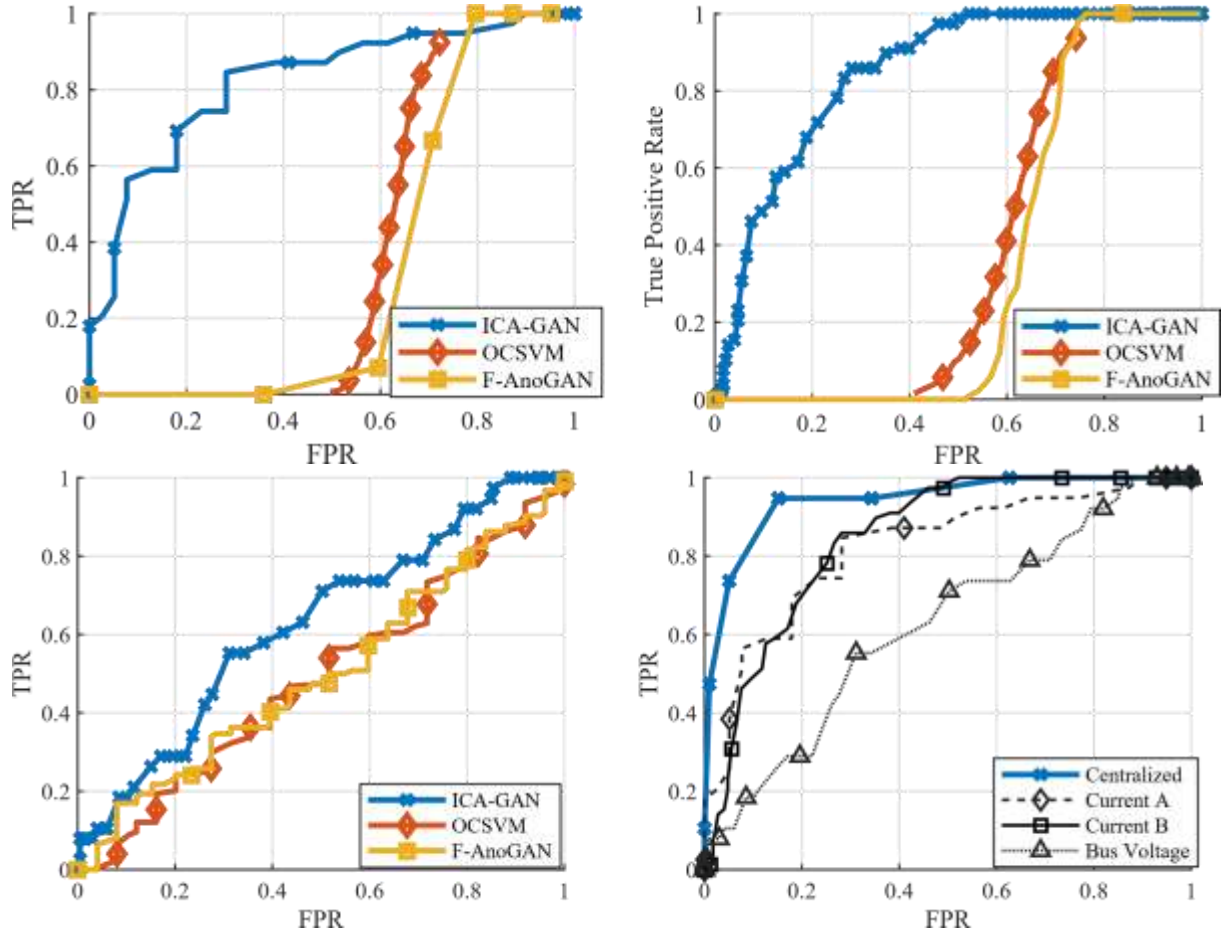


Figure 3.9 ROC curves (TPR at FRP=0.05). Top left: Detector at A (TPR=0.2564). Top right: Detector at B (TPR=0.3077). Bottom left: Detector at the bus (TPR=0.1053). Bottom right: Detector at the control center (TPR=0.7368.)

The detector at the control center combined two consecutive blocks of the three local decisions. Figure 3.9 shows the ROC curves of the local and central detectors¹¹. We observed that the central detector significantly improved the performance of local detectors, even when combining the less reliable bus voltage sensor. In particular, at $FPR = 0.05$, the TPR is above 0.7 whereas local detectors' TPR were below 0.31.

3.3.4.3 Case II: Natural Anomalies in PMU Measurements

Here we considered natural anomalies (bad data) involving multiple non-interacting anomalies. Two sets of simulations were performed. One is a small four-bus system used in the EPFL Smart Grid Project [58]. We used the 50 Hz PMU measurements collected on April 1st, 2016 from 5 PM to 6 PM. The second is based on the 133-bus synthetic North Texas transmission system [60] where one hour of 30 Hz PMU measurements are used in the simulation. We simulated non-

¹¹ We did not include OC-SVM and F-AnoGAN in the central detector performance because the local detectors for these algorithms did not have ROC curves above the diagonal to be useful.

interacting bad sequences and unobservable attack sequences for each system. For both systems, we used the one-phase equivalent of the three-phase systems.

The anomaly-free data were real-data measurements on the EPFL and North Texas Synthetic Systems. Gaussian mixture anomaly sequences were added to the anomaly-free measurements. Four of the ten measurements in the EPFL system and 6 out of 266 measurements in the North Texas system contained anomalies.

We separated the available data into training and test sets. Using EPFL data sequence we created a training set that has 1000 batches of 80 consecutive anomaly-free samples and a test set that has 500 batches of 80 anomaly sequences and 500 batches of 80 anomaly-free sequences for each measurement. Each test sequence consisted of 1.6 seconds of PMU measurement. Similarly, using the North Texas data, we created a training set with 900 batches of 80 consecutive anomaly-free samples from the historical samples, and a test set with 225 batches of 80 anomaly sequences and 225 batches of 80 anomaly-free sequences for each measurement. Each test sequence consisted of 2.6 seconds of PMU measurement.

The Wasserstein ICA-GAN was trained to obtain the transformation function from the measurements to the independent components. $b = 80$ consecutive measurements were used as inputs for the generator and the 50-dimensional output of the generator was transferred to the discriminator. We fed another 50-dimensional i.i.d. uniform samples to the discriminator.

As a preprocessing step before applying ICA-GAN, we used a linear least-squares prediction to decorrelate the measurement samples. The input layer of the ICA-GAN neural network was a linear least-squares predictor that whitens the input sequence.

After the ICA-GAN generator was used to convert the samples to *i.i.d.* sequence samples, we used an additional step to convert the distribution of the ICA sequence to uniform distribution. We used the empirical CDF of anomaly-free samples to achieve this transformation. After these steps, with the trained ICA-GAN we constructed the uniformity test algorithm. We used the samples to apply the K_1 -coincidence test as defined in (3.14). We used this approach for each measurement sequence individually. If at least one anomaly measurement sequence is detected, we assumed it is a successful detection.

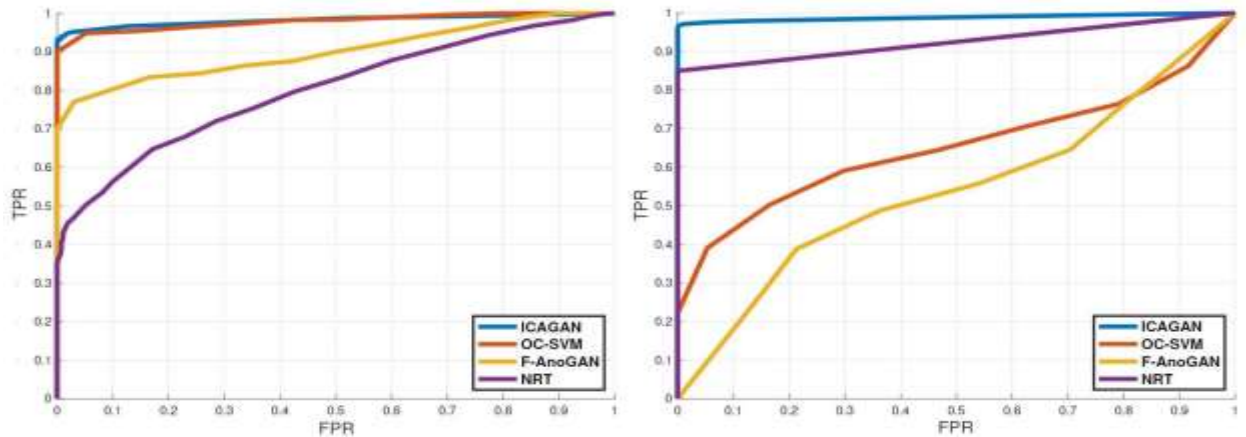


Figure 3.10 ROC curves for anomaly case 1. Left: The EPFL System. Right: The Synthetic North Texas system.

The ROC curves of ICA-GAN and benchmark techniques are in Figure 3.10. We observed that ICA-GAN achieved the best TPR across all FPRs and Table 3.3 (second column) shows TPR at FPR=0.05. ICA-GAN had a significantly higher true positive rate than the tested benchmarks.

Table 3.3 TPR values of different algorithms at FPR=0.05 constraint under bad data and data attack anomalies.

Algorithms	bad data (EPFL)	bad data (Texas)	data attack (EPFL)	data attack (Texas)
ICA-GAN	97%	95%	94%	80%
OC-SVM	94%	35%	61%	71%
F-AnoGAN	78%	5%	53%	18%
NRT	50%	88%	5%	5%

The conventional NRT did not work well on the EPFL system simulation compared to the Texas system simulation possibly because a larger ratio of measurements had a bad sequence. OC-SVM performed similarly to ICA-GAN's on the EPFL data set but was less successful on the Texas system simulation. F-AnoGAN had worse performance than ICA-GAN and OC-SVM in both cases. ICA-GAN had the best performance on both systems and had higher than 90% TPR even for small FPRs.

3.3.4.4 Case III: Unobservable Attacks on PMU State Estimation

We considered the extreme case of unobservable attack as an example to demonstrate the potential and importance of exploiting the inherent statistical properties and temporal dependencies of the data. An attack is "unobservable" when the data are manipulated so that the altered measurements and a fake state satisfy the underlying measurement equation. Therefore, no algebraic technique is capable of detecting such an attack. However, such attacks inevitably alter the underlying probability distribution and inter-temporal dependencies of the measurements. It is through these properties that our approach can make consistent detection.

We simulated the unobservable attacks, which is an extreme case of attack. The purpose was to illustrate that attacks that could not be detected based on the system model alone could be detected by ICA-GAN that exploited distribution properties. We constructed an unobservable data attack on the EPFL system and the North Texas system. We used the method in [45] to obtain an attack vector a_t . We assumed that the measurements in the system were such that the system was marginally observable. Then it was possible to design an unobservable data attack by manipulating 4 out of 10 measurements on EPFL system data and on 6 out of 266 measurements on North Texas synthetic system data. The attack vector can then be added to the measurements as an unobservable attack:

$$z_t' = z_t + w_t a_t,$$

where we constructed w_t from independent samples from a Gaussian Mixture Model.

Figure 3.11 showed the ROC curves where ICA-GAN had better performance than all compared methods with varying significance levels. The TPRs at FRP= 0.05 for the tested benchmarks were shown in the third column of Table 3.1.

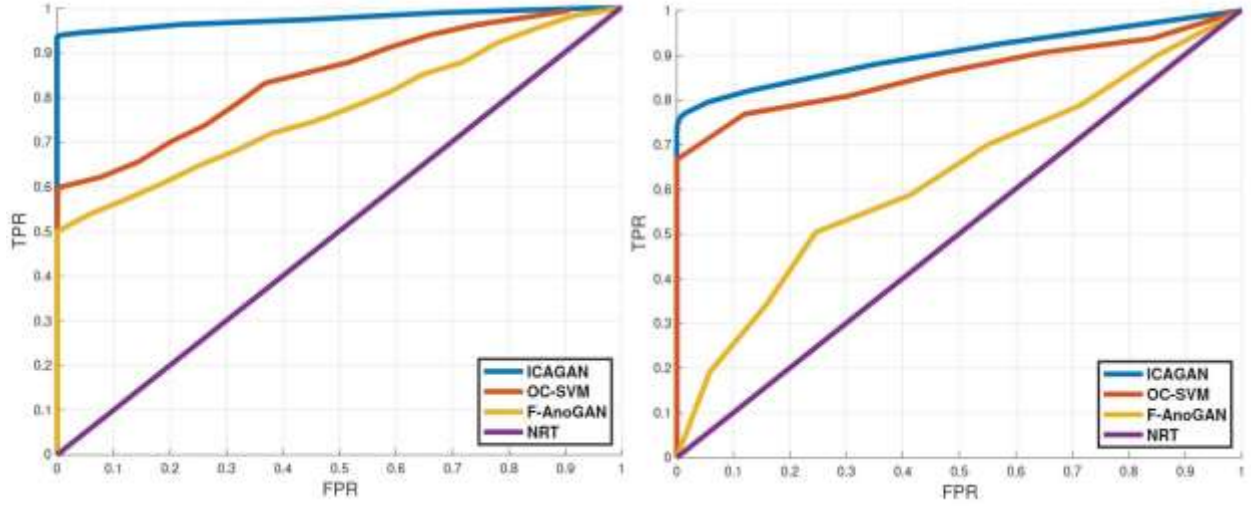


Figure 3.11 ROC curves for anomaly case 2. Left: The EPFL System. Right: The Synthetic North Texas system.

As expected, NRT performed as if it were a random selection without using measurements in both cases. OC-SVM performance was the closest to ICA-GAN, but there was still a significant difference. F-AnoGAN had a worse performance than OC-SVM on both systems. ICA-GAN had the best performance in both simulations because; i) the independent component analysis approach transformed the consecutive measurements to an independent sequence of samples ii) it detected the changes in the probability distribution rather than trying to find outlier samples.

Next, we experimented with the application of the anomaly detection scheme as a data cleansing step for state estimation. When the bad data was detected, we deleted the bad measurements from the measurement vector. A linear Bayesian estimator was used to replace anomaly sensor data with pseudo-measurements from clean measurements, followed by the standard weighted-least-squares state estimator.

Table 3.4 showed the average least-squares of the tested benchmarks along with the performance when there were no anomalies and the performance when anomalies were undetected. We observed that ICA-GAN had the potential as an effective data cleansing technique.

Table 3.4 Average squared error of state estimation.

Algorithms	bad data (EPFL)	bad data (Texas)	data attack (EPFL)	data attack (Texas)
Anomaly-free Meas.	6.6e-07	2.5 e-06	6.7e-07	4.3 e-06
Anomaly Meas.	1.1e-02	1.4 e-02	2.3e-02	2.3 e-02
Cleaned by ICA-GAN	1.7e-03	3.2 e-03	3.7e-03	6.4 e-03
Cleaned by OC-SVM	3.2e-03	6.2 e-03	1.5e-02	1.2 e-02
Cleaned by F-AnoGAN	6.1e-03	8.2 e-03	1.3e-02	2.1 e-02
Cleaned by NRT	8.7e-03	4.0 e-03	2.1e-02	2.2 e-02

3.3.5 Discussions

We discuss in this section some of the conceptual and implementation issues in the applications of the developed techniques.

3.3.5.1 Anomalies vs. System Dynamics

We make a practical (rather than mathematical) distinction between normal operations such as topology/load/generation changes from system or data anomalies. In practice, the proposed technique should be synchronized with real-time system operations so that the detection algorithm discounts measurements during the normal transient periods. How to coordinate system operations with anomaly detection in the monitoring system is of practical significance and deserves future investigation.

3.3.5.2 Anomalies in Training Data

We assume that training data are anomaly-free. Dealing with anomalies in training data is an active area of research beyond the scope of this work. In power system applications, it is reasonable to validate ex post facto whether the data are anomaly-free.

3.3.5.3 Offline vs. Online Training

A critical component of ICA-GAN is the GAN training of a neural network that extracts independent components. In principle, such training can be performed either offline using historical data or online using recent measurements. Effective online training, in particular, allows

the monitoring system to track system variations dynamically, provided that training converges quickly and training data validated as anomaly-free. In our experiments, 1.2 seconds of CPOW measurements appeared to be sufficient in the system anomaly detection in the EPFL battery energy system data set in Case Study I. For the relatively slower PMU measurements in Case Study II-III, 40 minutes data were used in training. These empirical results suggest that online training may potentially be viable.

3.3.6 Conclusion

We developed a data-driven deep learning approach to anomaly detection consists of sensor-level detectors that assume no prior models on anomaly and anomaly-free data; only anomaly-free training samples are used. Sensor-level decisions are combined at the control center to produce more reliable global decisions. To our best knowledge, the proposed technique is the first designed specifically for high-resolution measurements for CPOW and PMU streaming and can deal with both data and system anomalies.

4. Online Learning Distribution Voltage Controller with Convergence

4.1 Introduction to graph neural networks

To combat climate change, distributed energy resources (DERs) are rapidly being installed on the distribution grid. While DERs, such as rooftop photo-voltaic (PV) panels, provide affordable renewable energy they present new challenges for managing the voltage. Not only does the unpredictable nature of renewable generation adds a new layer of randomness due to its dependence on the weather, the distribution system was also not originally designed for the bi-directional power flow caused by DERs. Where in the past utilities have been able to passively regulate the voltage on the distribution system using seasonal adjustments [77] or rule-based methods, they are vulnerable when DERs are installed [78–80]. The control techniques designed for older versions of the grid are causing dangerous circuit conditions [81, 82], such as cascading outages and exploding equipment. Better solutions for safe operation are needed now.

More challenging is that due to the data resolution on the present system, the stochastic response cannot be ignored. Most solutions assume second or minute resolution [83–88] and disregard the stochastic proprieties of changing load, but this is far from the reality. Hourly measurements from limited feeder busses may be the only reliable source of information for most utilities. For example, the advanced metering infrastructure (AMI) [89], of our utility partners is designed almost exclusively to collect historical data for customer billing, and therefore has severe bandwidth bottlenecks even for hourly readings. The supervisory control and data acquisition (SCADA) [90] system has the capability to control regulators and capacitors, but these too record hourly readings from limited busses.

These data constraints align with utilities limited knowledge about their distribution system and result in limited situational awareness. Before the widespread use of DERs, the customer loading patterns ensured that power flow models were not necessary for regulation. So, while utilities are beginning to build the distribution feeder models, they are struggling due to the vastness of their systems and the frequent and unrecorded topology changes (especially on the secondary). Therefore, it is not currently feasible to rely on optimization methods that require accurate power flow models such as [91] and [92].

Though the distribution models, observability, and data resolution constraints will improve, it will be expensive and happen slowly. Since solutions are necessary now, we aim to regulate the distribution feeder voltage using the present system constraints (legacy device control with stochastic responses between measurements), scales as the data restrictions slowly improve, and provides convergence guarantees. These challenges can be overcome with model-free solutions using machine learning.

In the field of machine learning control, the rising star is Reinforcement Learning (RL). RL and its related fields are growing in popularity due to its ability to deal with unknown and dynamic learning environments [91]. Unfortunately, so far it has been impractical for researchers to transition RL to real world applications due to the experience needed to learn the optimal operation, the curse of dimensionality, and convergence guarantees [92]. A simpler solution would be to use

multi-armed bandits (MABs) which are the building blocks of RL and provide convergence guarantees.

While MABs learn quickly, their fundamental design assumes a system has only one state [93]. This makes MABs poorly suited for distribution operation since the system loading is dynamically changing sometimes unpredictably, which creates possibly infinite operating states. Contextual Bandits (CBs) are the middle ground between MABs and RL. By assuming an underlying relationship between contextual system information, CBs simplify RL but provide more flexible operation than MABs. Unfortunately, CBs are designed for online advertising and movie/article recommendations, so the theoretical results cannot provide guarantees about the sequential operation of distribution feeders [94, 95]. However, an underlying relationship exists between legacy devices and the system voltage, which could be used to learn a function approximator.

Designing a function approximator for distribution circuit regulation is a relatively new idea, as [85] and [86] recently proposed using machine learning regression using AMI data to determine control patterns of capacitors and voltage regulators. Unfortunately, they provide no guarantees about the performance and are implicit about the experience needed to learn accurate regression relationships. While these methods are designed for control, they do not build upon any kind of control theory which may be able to provide more basis for performance guarantees.

Other strategies implement gradient-based control in a model-free setting for distributed operation of DER devices on the distribution system in [87, 88]. These strategies provide performance guarantees, but gradient-based adaptive control is best for systems with continuous control actions and a slow-varying objective that may be stochastic. For these reasons they are a good choice for DERs. But adaptive control can be unpredictable and unstable [96] if not designed carefully, so it may be difficult or impossible to modify the ideas of the previous authors to operate legacy devices at hourly intervals. Unlike gradient-based adaptive control, self-tuning adaptive control learns the parameters directly. This strategy is similar to CBs and the regression techniques of [85] and [86], and would provide performance guarantees for sequential operation.

The self-tuning regulator (STR) is generally designed for linear systems with a stochastic noise component and was well studied in the 1970-80s [97–101]. This problem setting provides coordinated operation of devices with stochastic responses, does not suffer from the curse of dimensionality, and gives convergence guarantees for sequential operation. The analysis on a multi-output system [101] (to best reflect a distribution feeder with many voltage measurements) provides a strategy to learn the system parameters at the optimal rate. While the assumptions in the theoretical analysis may not perfectly transfer to distribution feeders, if the linear and stochastic assumptions from the general setting could be reconfigured a STR could be an effective solution.

We identify three main contributions for designing a data-driven method mimicking a STR for control of a distribution feeder's voltage. To coordinate the control of capacitors on a feeder when only hourly measurements are available, 1) we reconfigure the STR for distribution feeder voltage regulation. To overcome the linearity assumptions and improve the robustness 2) we design our system to easily embed the physical relationship of power and voltage and study the trade-off of using a linear representation and a quadratic one. Lastly, since model-free online learning control should have performance guarantees 3) we provide the analysis for asymptotic convergence. Our

approach is validated using Python to interface with OpenDSS [102] feeder models. We test on the IEEE 123-bus test feeder and a residential feeder from Duquense Light Company (DLC), the utility of the greater Pittsburgh, PA area.

4.2 Problem Setting

We aim to design an algorithm for utilities that have limited access to distribution feeder models, which we will refer to as a *model-free* setting. For the model-free setting in our paper, we make the following assumptions:

- 1) There is no power flow model of the distribution feeder and its corresponding impedance matrix.
- 2) There are at least a few network connected distributed feeder measurements which provide readings at hourly intervals.
- 3) There are distribution capacitors that are network connected and may be controlled from a central location.
 - a) Capacitors may be operated as continuous devices over a kvar range for experimental purposes.
 - b) Only the network connected distribution capacitors will be used to regulate the voltage.
- 4) The historical data is unavailable.
- 5) The only objective is to regulate the feeder voltage.

Assumptions 1 and 2 describe the realistic situation for many utilities according to the 2016 Department of Energy distribution system report [103]. While a few utilities may have slightly more capability than these 2 assumptions, many have much less. Assumptions 3, 3a, and 3b specify that we will only use reactive power adjustments to control the voltage in an experimental setting. The control strategies we use can be transferred to other devices that have step control. Assumption 4 specifies that no historical data is used since being reliant historical data could be a hindrance in cases where it is unavailable or biased. For example, when a load block is added/removed from a feeder or the capacitor settings have never/rarely changed. Assumption 5 establishes our sole objective in this paper. While there are other (and potentially competing) objectives when it comes to regulating a distribution feeder, it is easiest to understand when the objective is simple. The goal of the problem setting is that following our strategies the control of devices and the objectives can be adjusted for the needs and constraints of different model-free utilities.

To derive a data-driven model for our problem setting, we use the Newton-Raphson power flow relationship [104]. The Newton-Raphson method uses an iterative technique to describe how the voltage at a bus i will change based on the real and reactive power,

$$\begin{bmatrix} \Delta\delta_i \\ \Delta V_i \end{bmatrix} = \begin{bmatrix} \frac{\partial P_i}{\partial \delta_i} & \frac{\partial P_i}{\partial |V_i|} \\ \frac{\partial Q_i}{\partial \delta_i} & \frac{\partial Q_i}{\partial |V_i|} \end{bmatrix}^{-1} \begin{bmatrix} \Delta P_i \\ \Delta Q_i \end{bmatrix} \quad (4.1)$$

This equation can be simplified in a distribution circuit if we assume the change in voltage angle, $\Delta\delta$, has a minimal effect on the voltage magnitude. Also, since we are limiting ourselves to reactive

power control of capacitors, the change in active power, ΔP , is due only to the customer loading changing between the present and next state of the feeder. Since we cannot control this component, we consider the ΔP as part of the stochastic response of the system, which we detail in Sections 4.4 and 4.5. Therefore, the change in reactive power and the corresponding inverse Jacobian matrix determine the change in voltage such that,

$$\begin{bmatrix} \Delta V_1 \\ \vdots \\ \Delta V_k \end{bmatrix} = \begin{bmatrix} \frac{\partial Q_1}{\partial |V_1|} & \cdots & \frac{\partial Q_1}{\partial |V_k|} \\ \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial |V_1|} & \cdots & \frac{\partial Q_k}{\partial |V_k|} \end{bmatrix}^{-1} \begin{bmatrix} \Delta Q_1 \\ \vdots \\ \Delta Q_k \end{bmatrix}. \quad (4.2)$$

The inverse Jacobian describes the sensitivity matrix and is solved in an iterative manner with the Newton-Raphson method, since the values change based the system powers and voltages. But, in a distribution feeder when operating within the ANSI limits, the sensitivities may be approximated by a constant parameter matrix $A \in R^{k \times k}$ such that,

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,k} \\ \vdots & \ddots & \vdots \\ A_{k,1} & \cdots & A_{k,k} \end{bmatrix} \approx \begin{bmatrix} \frac{\partial Q_1}{\partial |V_1|} & \cdots & \frac{\partial Q_1}{\partial |V_k|} \\ \vdots & \ddots & \vdots \\ \frac{\partial Q_k}{\partial |V_1|} & \cdots & \frac{\partial Q_k}{\partial |V_k|} \end{bmatrix}^{-1}. \quad (4.3)$$

Using data-driven methods the parameter matrix A may be determined, and the bus voltages based on the reactive power changes can be estimated. Note that $\Delta Q = Q(t+1) - Q(t)$, and $\Delta V = V(t+1) - V(t)$, so we can expand the approximation such that,

$$\begin{bmatrix} V_1(t+1) \\ \vdots \\ V_k(t+1) \end{bmatrix} \approx A \begin{bmatrix} Q_1(t+1) \\ \vdots \\ Q_k(t+1) \end{bmatrix} - A \begin{bmatrix} Q_1(t) \\ \vdots \\ Q_k(t) \end{bmatrix} + \begin{bmatrix} V_1(t) \\ \vdots \\ V_k(t) \end{bmatrix}. \quad (4.4)$$

We use this approximation to derive control strategies for distribution feeders in the remainder of the paper.

4.3 Linear Voltage Controller

Based on the approximate relationship in Eq. 4 we aim to design an equality that can learn the power flow relationship. Since the voltages are the regulated outputs, we define them at stage n with the column vector $y_n = (V_1(n), \dots, V_k(n))^T$. Likewise, the control variable is the reactive power at each capacitor described by the column vector $u_n = (Q_1(n), \dots, Q_r(n))^T$. To transition from an approximation to an equality, we introduce the column vector $\varepsilon_n \in R^k$ as a stochastic component describing the approximation error, the error from the measurement devices, and the

unpredictability of loading changing between measurement intervals. Using these definitions, we rewrite (4.4) in a general format that can be learned as,

$$\mathbf{y}_n = \mathbf{c} \odot \mathbf{y}_{n-1} + D\mathbf{u}_{n-1} + B\mathbf{u}_n + \varepsilon_n, \quad (4.5)$$

where $\mathbf{c} \in R^k$, $D \in R^{k \times r}$, and $B \in R^{k \times r}$ are the learned parameters if r is the number of regulated capacitors. Note that $\mathbf{c} \odot \mathbf{y}_{n-1}$ is the element-wise multiplication between vectors.

(4.5) may also be written as a set of single-output equation, which clarifies how the parameters are learned. Defining the measured busses as $i \in (1, 2, \dots, k)$, the voltage at stage n becomes $y_{n,i}$. The parameters that affect the voltage are c_i , $\mathbf{d}_i \in R^r$, and $\mathbf{b}_i \in R^r$, which are the values of the respective rows of the parameters in (4.5). We contain the parameters in vector $\boldsymbol{\varphi}_i = (c_i, \mathbf{d}_i, \mathbf{b}_i)^T$, and the inputs by $\mathbf{w}_i(n) = (y_{n-1,i}, \mathbf{u}_{n-1}, \mathbf{u}_n)^T$, so that we can define the system as a set of linear equations such that,

$$y_{n,i} = c_i y_{n-1,i} + \mathbf{d}_i^T \mathbf{u}_{n-1} + \mathbf{b}_i^T \mathbf{u}_n + \varepsilon_{n,i} \quad (4.6)$$

$$= \boldsymbol{\varphi}_i^T \mathbf{w}_i + \varepsilon_{n,i} \quad (4.7)$$

Since the parameters $\boldsymbol{\varphi}_i$ are initially unknown they can be estimated with least squares estimation (LSE). To mathematically write the solution to LSE, we let $W_{i,n} = [\mathbf{w}_{i,1}, \dots, \mathbf{w}_{i,n}]^T \in R^{n \times 1+2r}$ be the matrix recording the inputs at for a bus i at each stage. Similarly, the column vector $\mathbf{y}_{i,n}$ of length n records the voltage outputs at bus i . Naturally, the LSE of the parameters at stage n is determined such that,

$$\widehat{\boldsymbol{\varphi}}_i(n) = (W_{i,n}^T W_{i,n})^{-1} W_{i,n}^T \mathbf{y}_{i,n}. \quad (4.8)$$

After each stage, a new row of inputs are added to each W_i and a new output to $\mathbf{y}_i$. So, once there is a LSE solution $\widehat{\boldsymbol{\varphi}}_i(n)$ can be determined anew at each stage.

The learned parameters are most accurate when the input samples of $W_{i,n}$ are chosen at random. Unfortunately, this is not practical since the primary objective is to regulate the voltage on the distribution feeder. But some random operation is necessary, especially in the initial stages before there is a solution to the LSE, i.e. $(W_{i,n}^T W_{i,n})$ is not invertible. Therefore, we introduce the measure of *regret* to highlight the price of learning the parameters and quantifies the effectiveness of regulating the voltage.

Prior to mathematically defining the regret, we need to introduce the idea of a policy π . There is one unique optimal policy π^* , but any policy that chooses regulator settings (i.e., random, seasonal, constant, etc.) belongs to the policy set $\pi \in \Pi$. The capacitor inputs $\mathbf{u}_n^\pi$ are chosen according to a policy's rules, and result in output voltages $\mathbf{y}_n^\pi$. The optimal policy π^* chooses the capacitor inputs $\mathbf{u}_n$ so that the resulting voltages $\mathbf{y}_n^*$ are as close as possible to the nominal voltages

y^* of the measured busses, which could be set to any value but we have them set to 1.0 volts per-unit. In scenarios when there are more measured busses than capacitors $k > r$, it may not be possible for the optimal policy to choose settings where the voltages at the measured busses are nominal. Therefore, we define the *instantaneous* regret at stage n as the difference between the L2 norm of the voltage deviation between policy π and the optimal policy, such that,

$$r^\pi(n) = (\mathbf{y}_n^\pi - \mathbf{y}^*)^\top (\mathbf{y}_n^\pi - \mathbf{y}^*) - (\mathbf{y}_n^* - \mathbf{y}^*)^\top (\mathbf{y}_n^* - \mathbf{y}^*). \quad (4.9)$$

The regret quantifies how a policy regulates the measured outputs over time, which is the running sum of the instantaneous regret, such that

$$R^\pi(n) = \sum_{t=1}^n r^\pi(t). \quad (4.10)$$

The regret quantifies how well a policy regulates the voltage at the *measured* busses. What if we want to evaluate how well the system, including the non-measured busses, is being regulated? We can do in our experiments, but need to introduce the system regret $R_s(n)$ as a separate metric. So, the system voltages at stage n are defined by γ_n . Note that the measured bus voltages are a subset of the system voltages, $\mathbf{y}_n \subseteq \gamma_n$. The optimal policy for the system regulation therefore may be different than the optimal policy for observable regulation, so we define the instantaneous *system* regret of a policy at stage n as,

$$r_s^\pi(n) = (\gamma_n^\pi - \gamma^*)^\top (\gamma_n^\pi - \gamma^*) - (\gamma_n^* - \gamma^*)^\top (\gamma_n^* - \gamma^*). \quad (4.11)$$

The resulting system regret of a policy naturally becomes,

$$R_s^\pi(n) = \sum_{t=1}^n r_s^\pi(t). \quad (4.12)$$

The system regret is only useful in experimental settings to compare how well a feeder was regulated for different configurations of observability. Also note that in a real-world model-free setting, the optimal policy is unknown and therefore the regret of (4.12) cannot be calculated. These metrics are used in our experiments to show how close our algorithm gets to optimal voltage regulation.

4.4 Asymptotic Convergence Guarantees

The previous sections introduced the structure of the models to estimate the output voltages $\mathbf{y}$. Once converged, these equations are used to choose the capacitor states $\mathbf{u}$ to keep the growth of the regret to a minimal (4.12). However, we have yet to specifically define a policy to learn the parameters and regulate the system. So, in this section, we introduce a policy to ensure the

parameters are learned but the voltage is regulated as best as possible, provide the necessary conditions for asymptotic convergence, and introduce theorems for performance guarantees.

4.4.1 Policy for Asymptotic Convergence of Linear Self-Tuning Regulators

A policy chooses how to set the states of the regulators. In online learning methods this raises the dilemma of when to explore for information gain (take a random action) or exploit for best expected performance i.e., determine $\mathbf{u}_n$ that will minimize the regret based on the LSEs. Our policy will exploit all the time, except for at the stages in set I , which is defined as,

$$I = \{1, \dots, \tau^*\} \cup \{\tau^* + v_1, \tau^* + v_2, \dots\}. \quad (4.13)$$

The set I is the union of 2 sets, initialization and continued exploration. The initialization set, $\{1, \dots, \tau^*\}$, establishes that during the first τ^* stages the policy will randomly set the capacitor states $\mathbf{u}_n$. The continued exploration set, $\{\tau^* + v_1, \tau^* + v_2, \dots\}$, ensures that as $n \rightarrow \infty$ the system will operate the capacitors randomly infinitely many times. To understand how to choose τ^* and the values of v , we look into the conditions for strongly consistent LSEs.

4.4.2 Conditions for Strongly Consistent Least Square Estimates

For the LSEs to approach their converged values, the estimates must remain strongly consistent. There are many conditions that must be met for strongly consistent LSEs [97,98,101,108,109], but 3 of them are relevant to power systems. The first condition requires that the stochastic component is not biased and the variance is finite. The second condition is that the regulator variable $\mathbf{u}_n$ should be set randomly until there is a solution to the least squares. Lastly, there are requirements for the minimum and maximum eigenvalues of the invertible square matrix in the least squares solution. We use the notation for the set of single-output voltage estimates to define the 3 conditions for strongly consistent LSEs.

Condition 1. The stochastic component must satisfy bias and variance requirements, specifically,

$$\mathbb{E}(\varepsilon_{n,i}) = 0, \quad (4.14)$$

$$\sup_n \mathbb{E}(\|\varepsilon_{n,i}\|^\alpha) < \infty \text{ for some } \alpha > 2. \quad (4.15)$$

In an optimal problem setting, the optimal regret accumulation rate is $\sigma^2 \log(n)$, where σ^2 is the variance of the stochastic component ε . This problem setting is considered optimal because it establishes that $\mathbb{E}(\varepsilon) = 0$. While this is convenient for mathematicians to provide theoretical guarantees, it is not very practical. For example, the expected value of the stochastic component for each bus i is based on the measurement device accuracy (measurement error m_i), samples used for data-driven approximation (modeling error l_i), and users load consumption (feedback delay f_i). These values can be studied separately and their summation completes the stochastic component such that,

$$\varepsilon_i = m_i + l_i + f_i, \quad (4.16)$$

$$\mathbb{E}(\varepsilon_i) = \mathbb{E}(m_i) = \mathbb{E}(l_i) = \mathbb{E}(f_i) = 0 \quad (4.17)$$

These stochastic and error components may never be proven to have an expected value exactly at 0. However, we do expect them to be close.

Previous work generally assumes the measurement error m in distribution feeders to be unbiased and Gaussian. More recent studies have shown that a combined Gaussian may be a better assumption, but that the expected error is still nearly 0 [110]. The second component, the feedback delay f is an outcome of the load unpredictably changing in-between stages $n - 1$ and n . Previous work has shown this component to be Gaussian for shorter intervals [111], and these results intuitively make sense due to the cyclic nature of load consumption. We examined this claim using the NREL loading data-set integrated on the DLC feeder and found similar results. Shown in Figure 4.1, the empirical results of the voltage changing at one bus with a corresponding fitted Gaussian has an expected value of nearly 0. Though the x-axis plotted is based on a 13.279 KV-LL base, the determined average change in voltage between hourly measurements is only 0.036 volts, which is $\frac{0.036}{200} \approx .02\%$ of the maximum change.

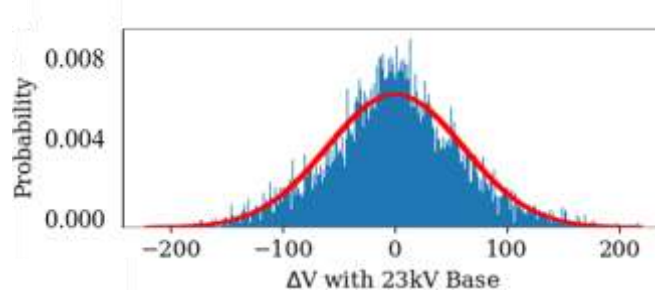


Figure 4.1: The change in voltage data from a year of unregulated hourly operation at a measured bus. Fitted with a Gaussian based on the expected value and variance of the data, $\mathbb{E}(\Delta V) = 0.036$ and $\sigma^2 = 3636.62$.

The third error component l arises from assuming constants for the sensitivity matrix and using LSE to determine their approximate values. Since these constants are found using data-driven regression, l would be unbiased if the control of $\mathbf{u}_n$ is chosen randomly. While we only randomly operate in set I , Lemma 1 and Lemma 2 from [101] prove that random operation is not necessary all the time for the values to converge. Further, the empirical results from our simulations are very promising and suggest that the variance of this component is minimal when using the linear representation of (4.6). This is consistent with other simplified power flow solutions [105], [106] and is highlighted in our experiments in Section 4.5.

To summarize, while we cannot claim to meet an optimal regret rate, it would be impossible in the problem setting detailed due to the sources of unpredictability and error in a distribution feeder.

Fortunately, the stochastic components are small compared to the control range and there is justification in previous works and our own that the variance is finite and the expected value is close to 0 to nearly satisfy Condition 1.

Condition 2. The capacitor states must be assigned randomly until $(W_{i,n}^T W_{i,n})$ is invertible.

Condition 3. The minimum eigenvalue (λ_{min}) of the square matrix approaches infinity, and faster than the log of the maximum eigenvalue $\log(\lambda_{max})$, such that

$$\lambda_{min}(W_{i,n}^T W_{i,n}) \rightarrow \infty \text{ a. s.}, \quad (4.18)$$

$$\log\left(\lambda_{max}(W_{i,n}^T W_{i,n})\right)/\lambda_{min}(W_{i,n}^T W_{i,n}) \rightarrow 0 \text{ a. s.} \quad (4.19)$$

These conditions provide the basis to introduce the theorems for performance guarantees which guide the design of the self-tuning regulator policy.

4.4.3 Ensuring Asymptotic Convergence

The conditions for strongly consistent LSEs lead to 3 straightforward theorems that ensure the initial linear estimate is stable, the continued operation stays strongly consistent, and the guarantees are transferable to a non-linear space using kernels.

Theorem 1. *If the stochastic components satisfy Condition 1 and operation is random until $(W_{i,n}^T W_{i,n})$ is invertible, then the LSE for bus i at stage n is strongly consistent, from Lemma 1 [101].*

This stems from conditions 1 and 2. Using this theorem we define τ_i as the minimum number of stages for random operation until the LSEs of bus i are strongly consistent such that,

$$\tau_i = \inf_n \left\{ \lambda_{min}(W_{i,n}^T W_{i,n}) > 0 \right\}. \quad (4.20)$$

Following this strategy, τ_i is defined when the LSE of all the busses have been defined, such that

$$\tau^* = \max_i (\tau_i) \quad (4.21)$$

Theorem 2. *If $(W_{i,n}^T W_{i,n})$ is invertible and Condition 1 is satisfied, randomly setting the capacitor states need only occur at a rate that is less than $o(\log n)^\delta$ for some $\delta > 1$, from Theorem 3 [101].*

This stems from the third condition and therefore we define the integers for continued exploration such that

$$v_j = \exp\{(1 + o(1))j^\delta\} \text{ for some } \delta > 1. \quad (4.22)$$

The random operation after the initial stable estimate occurs at a rate exponentially less than $O(\log n)$ and therefore contributes negligibly to the regret as $n \rightarrow \infty$.

Theorem 3. If Condition 1 is satisfied, (4.6) can be lifted to the quadratic kernel space. Since the function approximator is linear in the new domain, Theorem 1 and Theorem 2 do not change.

When the system is non-linear a kernel can transform the input space to fit a linear representation in higher dimensions. Different kernel methods can be used for a variety of non-linear shapes, but the homogeneous quadratic kernel captures the relationship between power and voltage. When the system is fully observable, the quadratic kernel will exactly represent the physical law-based power flow [106]. Therefore, the variance of the modeling error l is reduced compared to the linear estimate while m and f remain unchanged.

Theorem 1 and 2 ensure the regret from exploration is negligible as $n \rightarrow \infty$, and Theorem 3 ensures those theorems can be applied when using the homogeneous quadratic kernel. Therefore, the regret of our policy will be close to approaching the regret rate of $\sigma^2 \log n$, where σ^2 represents the variance of the combination of stochastic components ε . The algorithm to implement this policy using the linear representation of (4.6) is shown in Algorithm 1. The algorithm can be easily modified for the homogeneous quadratic representation of Eq. 15 by fitting the homogeneous quadratic parameters φ_i^h to their respective inputs $\phi_h(\mathbf{w}_i(\mathbf{n}))$.

Algorithm 1: Linear Self-Tuning Regulator for Distribution Voltage

Result: Regulate the circuit voltage without initially knowing the system parameters

```

1  $\tau^* \leftarrow -1$ 
2  $\delta > 1$ 
3  $v = \{\exp(1^\delta), \exp(2^\delta), \dots\}$ 
4 Let  $W_i$  be the recorded inputs of bus  $i$  voltage estimate
5 Let  $z_i$  be the recorded output voltages at bus  $i$ 
6  $I \leftarrow$  empty set
7 for  $n \in (1, \dots, \infty)$  do
8   Collect the capacitor states  $u_{n-1}$ 
9   Extract the most recent voltage measurements  $y_{n-1}$ 
10  if any  $\left( \lambda_{\min}(W_{i,n}^\top W_{i,n}) \leq 0 \right)$  or  $n \in I$  then
11     $u_n \leftarrow$  randomly generated
12    if  $\tau^* = -1$  then
13       $\tau^* \leftarrow n$ 
14       $I = \{1, \dots, \tau^*\} \cup \{\tau^* + v_1, \tau^* + v_2, \dots\}$ 
15    end
16  else
17     $u_n \leftarrow \arg \min_u \left( \hat{c} \circ y_{n-1} + \hat{D}u_{n-1} + \hat{B}u_n \right)$ 
18  end
19  Adjust feeder capacitors to  $u_n$ 
20  Wait for circuit feedback  $y_n$ 
21  for  $i \in (1, \dots, k)$  do
22    append  $(y_{i,n-1}, u_{n-1}, u_n)$  to  $W_i$ 
23    append  $y_{i,n}$  to  $z_i$ 
24     $\{\hat{c}_i; \hat{d}_i; \hat{b}_i\} \leftarrow z_i^\top W_i (W_i^\top W_i)^{-1}$ 
25  end
26 end

```

4.5 Empirical Validation

In this section, we perform extensive simulations using Python to implement tests and interact with OpenDSS distribution feeder models via the COM interface. The feeder models include the standard IEEE 123-bus and a residential feeder from DLC, shown in Figure 4.2. The time series voltages were generated in OpenDSS using load profiles from the NREL synthetic dataset [112]. The load profiles magnitudes were adjusted to provide volatile feeder loading so that optimal regulation required the capacitors to change states. Our goal is to 1) verify the convergence guarantees of the linear and quadratic models, 2) compare converged performance, and 3) examine scaling properties.

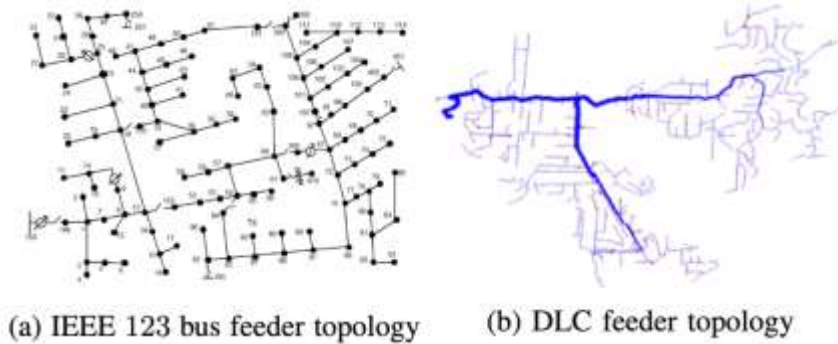


Figure 4.2 The feeders used for our experiments.

4.5.1 Testing Asymptotic Convergence

To highlight the slight differences between implementing Algorithm 1 using the linear (4.7) representation, we eliminate the measurement and feedback delay ($\mathbf{m} = \mathbf{f} = \mathbf{0}$) and the capacitors are operated as analog devices i.e., over a kvar range rather than on/off. To determine the optimal kvar values of the capacitors at each stage, $\mathbf{u}_k$, the Nelder-Mead minimization algorithm [113] was implemented with the SciPy minimization function in Python to directly interact with the OpenDSS model. The average regret, $R(n)/n$, after nearly a year of hourly operations on the IEEE 123 bus test feeder with 4 capacitors, show the regret converging to a nearly 0 in Figure 4.3.

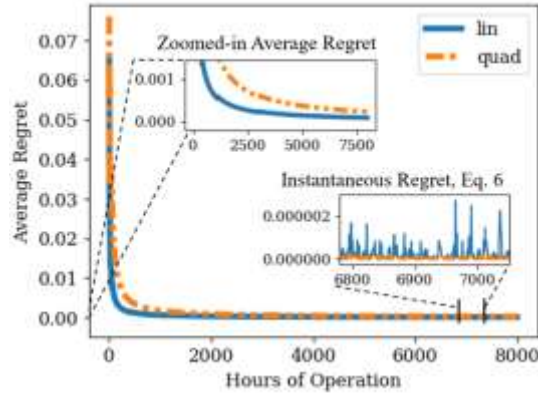


Figure 4.3 The overarching figure shows the average regret of the linear and quadratic models *without* feedback delay. The smaller zoomed-in average regret shows that the linear model has a lower average after almost a year of hourly operation. On the contrary, the instantaneous regret snapshot shows that once converged, the quadratic model, regulates the system better.

From Figure 4.3, we can see that both the linear and quadratic representations converge while the average regret of the linear model is slightly lower at the end of the simulation. This is due to the homogeneous quadratic kernel initialization period that took 45 stages to determine τ^* , compared to 9 stages when using the linear model. The snippet from the instantaneous regret (4.9), shows that the converged operation of the quadratic model is more precise than the linear version at a very small scale. Also, this highlights the capability of the homogeneous quadratic kernel to essentially eliminate the error component l . So even though the average regret of the quadratic model is higher after 8000 simulated hours, it would surpass the linear version given enough time.

To mimic realistic operation, we retain the hourly feedback delay $\mathbf{f}$ in Fig 4.4. Even with the additional stochastic component, both representations converge even though it increases the instantaneous regret (highlighted in the instantaneous regret snapshot). Since the feedback delay noise accounts for roughly 3 magnitudes more error than the linear delay (i.e., $l \times 10^3 \propto \mathbf{f}$) the advantage of using the quadratic representation is reduced.

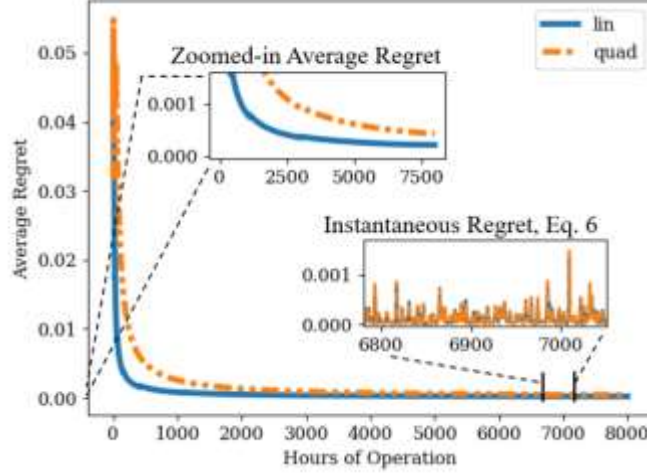


Figure 4.4 Similar to Figure 4.3, the overarching figure shows the average regret of the linear and quadratic models.

4.5.2 Evaluating Converged Regulation

In the previous subsection our experiments showed the quadratic model regulated the system slightly better when there was no feedback delay. In this subsection we expand on our examination of the converged operation by determining the coefficient of determination. More commonly known as the R^2 score, the coefficient of determination measures the residual error proportional to the variance of the system outputs. By letting the mean of the voltages observed at a bus be $\bar{y}_i = \frac{1}{n} \sum_{t=1}^n y_{t,i}$ and the predicted output as $\hat{y}_{n,1}$, the R^2 score for a voltage estimate of bus i becomes,

$$R_i^2 = 1 - \frac{\sum_{t=1}^n (y_{t,i} - \hat{y}_{t,i})^2}{\sum_{t=1}^n (y_{t,i} - \bar{y}_i)^2}. \quad (4.23)$$

A regression model that perfectly predicts the bus voltage would result in $R_i^2 = 1$. So, to compare the different regression models, we average the R_i^2 scores for each bus in an extreme test where distribution capacitors may be set to any kvar within a range that can alter the voltage well past the ANSI limits. To produce data-rich converged regression models, they are trained with random loading points without feedback delay and capacitor operations. We determine the average R^2 scores at a constant system loading by testing different random capacitor operations. The system loading is altered to provide an x-axis for Figure 4.5, which shows the average R^2 scores for the DLC and IEEE 123 bus feeders.

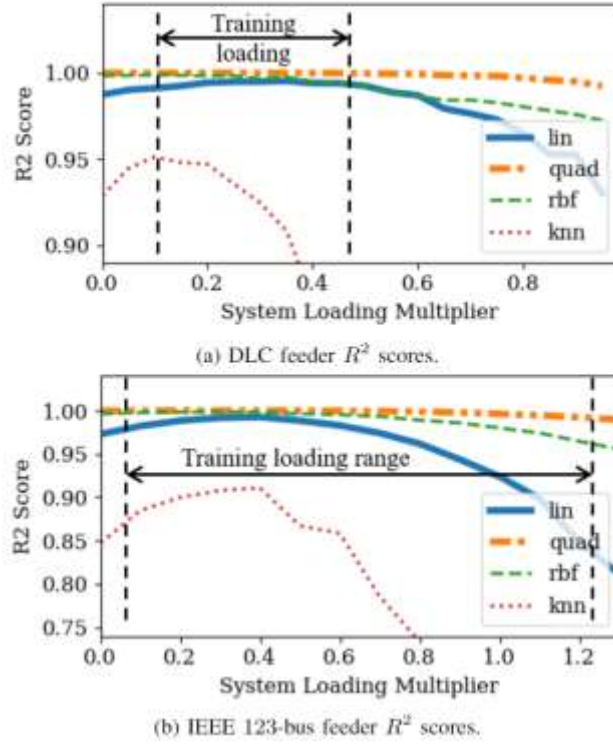


Figure 4.5 The R^2 scores for the DLC (a) and IEEE 123-bus (b) feeder based on the system loading when estimating the voltage.

The quadratic kernel is clearly the most robust, since it naturally captures the quadratic shape of the power flow relationship. The model using the RBF kernel is next best due to the large number of features. The linear model behaves best when the system loading is normal, while the KNN is inferior since there is no feature selection.

4.5.3 Comparing Scaling Properties of the Regulators

The previous subsections showed the linear model with a lower average regret after a long period of operation, but the quadratic model being more precise once converged in cases without feedback delay. A third dimension that is important to understand is as the distribution system is modernized, is how the models performance will change as the observability (number of measured busses) and controllability (number of network connected regulation devices) of the feeder improves.

4.5.3.1 Examining How Observability Affects System Regulation

We examine how the initialization period changes based on the observability, i.e., the number of observable busses. To evaluate the performance as the observability improves, we consider the IEEE 123 bus system regret average, $R_s(n)/n$. We simulate for 250 stages which highlights the effect of initial random operation on the regret but also shows the improvement of the system regulation. The results of changing the number of measured busses effect on the system regret is shown in Figure 4.6.

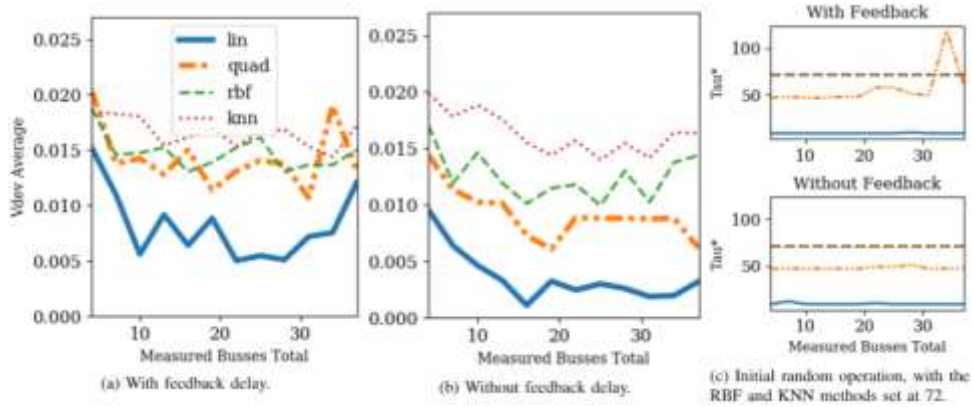


Figure 4.6 Figures (a) and (b) show the final average voltage deviation from 250 stages on the IEEE 123-bus feeder, according to the number of observable busses. Figure (c) shows the number stages of initial random exploration require to establish a stable estimate.

The linear design with our policy results in the least voltage deviation across the board and decreases as the observability improves. More importantly the initialization of both the linear and quadratic remains relatively constant except for the spike in the quadratic representation in Figure 4.6c, which occurs due an observed bus that is less sensitive to capacitor operations and the randomness of the initial capacitor operations. From the results in Figure 4.6c we can conclude that observing more busses is unlikely to increase the initialization.

Observing the results when there is no feedback delay, Figure 4.6b, the advantage of using more busses begins to flatten out at about 15 observable busses, or 10-15% of the busses in the IEEE 123-bus feeder. At this point, it becomes unnecessary to observe more busses to improve the overall system regulation.

4.5.3.2 How Controllability Affects Regulation

Unlike the observability, the initial performance is heavily dependent on the number of capacitors being regulated in the system. So, to visualize the result of Condition 2, we plot the minimum initial random operations (i.e., minimum value of τ^* from (4.21)) versus the numbers of capacitors, in Fig. 4.7.

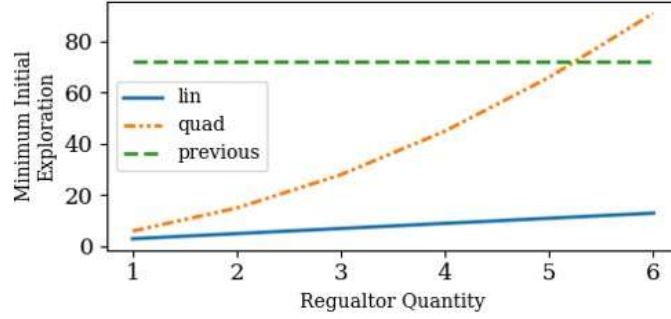
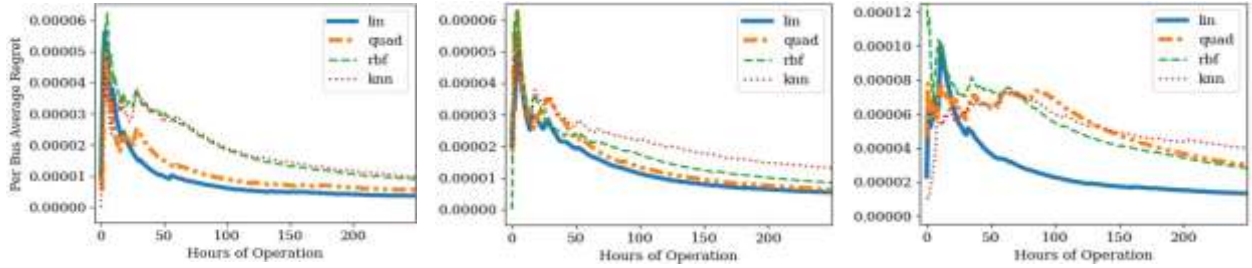


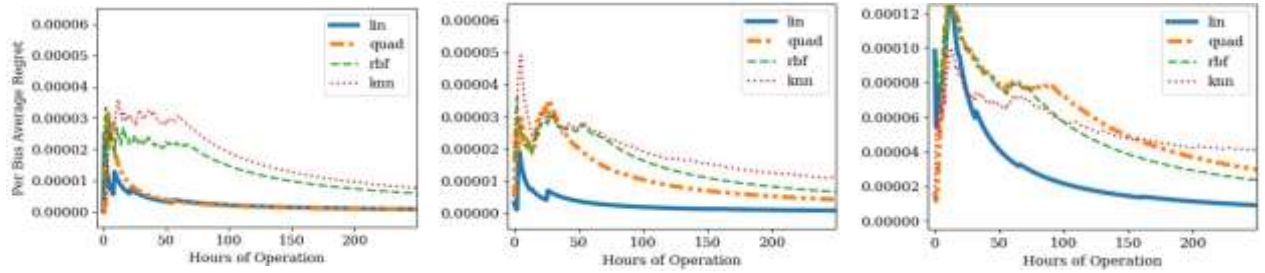
Figure 4.7 Minimum values of τ^* , where *lin* refers to the linear model (4.6), *quad* refers to the homogeneous quadratic model, and *previous* refers to 3 days worth of initial exploration roughly recommended by the previous methods [85], [86].

From Figure 4.7 it is clear that the quadratic model increases exponentially based on the number controlled capacitors, while the linear model increases linearly. The previous methods of [85], [86], are set to explore for 3 days (72 stages) but do not guarantee accurately converged models. To see how the number of capacitors affects the regret we highlight the average regret normalized for the number of observable busses, $(R(n)/n)/k$, in Figure 4.8 and Figure 4.9, with and without feedback delay (f), respectively.



(a) Controlling 1 capacitor. (b) Controlling 3 capacitors. (c) Controlling 6 capacitors.

Figure 4.8 The normalized average regret, $(R(n)/n)/k$, for the DLC feeder with hourly feedback delay.



(a) Controlling 1 capacitor. (b) Controlling 3 capacitors. (c) Controlling 6 capacitors.

Figure 4.9 The normalized average regret, $(R(n)/n)/k$, for the DLC feeder without feedback delay.

In all cases, the linear model has the lowest average regret since it learns the fastest and the number of features increase at a linear rate. On the other hand, the average regret of the quadratic model suffers as the number of capacitors increases since the minimum initial random operation grows exponentially according

In all cases, the linear model has the lowest average regret since it learns the fastest and the number of features increase at a linear rate. On the other hand, the average regret of the quadratic model suffers as the number of capacitors increases since the minimum initial random operation grows exponentially according to $\binom{2+2r}{2}$, where r is the number of controlled capacitors.

4.6 Conclusion and Future Work

We provided an algorithm to regulate the voltage on a distribution system in a model-free setting. The linear model prioritized simplicity and low initial learning time, while the quadratic version valued precision and robustness. Unlike similar previous work, we provided provable performance guarantees. The choice of which regulator to use depends on the feeder parameters and the system operators' preferences.

Future work will involve improving our algorithm to provide performance guarantees when using intelligent (not purely random) exploration to avoid dangerous circuit conditions. Also, we are looking to make our algorithm adaptive by integrating the capability to recognize if the topology has changed based on the system responses.

5. Conclusion and Future Work

In summary, the method on SNR standard deviation analysis in PSERC project on S-74 was extended to (1) the use with additional asset failure cases, (2) the use of threshold-based asset degradation detection methods was investigated on a wider range of substation PMU sources, (3) the use of support vector classification (SVC) to classify asset state-of-health was tested and automated, (4) the use of the Matrix Profile technique in detecting anomalous power system behavior, and finally, (5) the use of derivative difference matrices to map asset degradation locations was implemented. Of these techniques, the SVC algorithm developed here proved to be a promising asset health indicator as evident in its accuracy of over 96% and over 99% on two different real-world test data sets. The derivative difference matrix computation method also proved to be a fast and computationally efficient method for detecting anomalous behavior in large number of concurrent data streams. In the future, further refinement of the SVC algorithm as well as automation of the interpretation of the derivative difference matrix results will enable greater visibility into the behavior and performance of the electric power systems of those who leverage these techniques. It is hoped that through the use of AI, electric power stakeholders will benefit from greater system reliability and safety.

For monitoring, this project developed a data-driven deep learning approach to anomaly detection consists of sensor-level detectors that assume no prior models on anomaly and anomaly-free data; only anomaly-free training samples are used. Sensor-level decisions are combined at the control center to produce more reliable global decisions. To our best knowledge, the proposed technique is the first designed specifically for high-resolution measurements for CPOW and PMU streaming and can deal with both data and system anomalies.

Finally, we provided an algorithm to regulate the voltage on a distribution system in a model-free setting. The linear model prioritized simplicity and low initial learning time, while the quadratic version valued precision and robustness. Unlike similar previous work, we provided provable performance guarantees. The choice of which regulator to use depends on the feeder parameters and the system operators' preferences. Future work will involve improving our algorithm to provide performance guarantees when using intelligent (not purely random) exploration to avoid dangerous circuit conditions. Also, we are looking to make our algorithm adaptive by integrating the capability to recognize if the topology has changed based on the system responses.

Appendix A: SNR_STD_exporter.py

```
# This program calculates the SNR standard deviation of each
# voltage and current phase and exports them to files where
# they can be used by a subsequent process.
# Plots are also produced for the original, SNR, and SNR STD plots
# of each phase.
# Please specify the file to be processed in the variable "sourceFile".
# The input file should be in CSV format.
# Please specify the column number of each relevant data column
# using the variables "VAM_col" through "ICM_col".
# -----Imports-----
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import gc
# -----Functions-----
# SNR function (with zero-SNR and dropouts removed):
def SNR_Calculation_DM_ZeroSTD_Removal(inp, Window_Size_SNR):
    outp = np.zeros(len(inp))
    for i in range(len(inp)):
        if i <= (Window_Size_SNR-1):
            outp[i] = 0
        if i > (Window_Size_SNR-1):
            test = inp[(i-(Window_Size_SNR-1)):i]
            if (len(test[test == 0])) == 0:
                computed_std = np.std(test)
                if computed_std == 0:
                    outp[i] = np.NaN
                else:
                    outp[i] = 10*np.log10(np.mean(test)/computed_std)
            else:
                outp[i] = np.NaN
    return outp
# Moving window standard deviation function:
def stddev(inp, Window_Size_STD):
    outp = np.zeros(len(inp))
    for i in range(len(inp)):
        if i <= (Window_Size_STD-1):
            outp[i] = 0
        if i > (Window_Size_STD-1):
            test = inp[(i-(Window_Size_STD-1)):i]
            outp[i] = np.std(test)
    return outp
# SNR and SNR standard deviation plotter and exporter function.
# SNR standard deviation series is returned.
def exportSNRSTD(data, Window_Size_SNR, Window_Size_STD, origFile, SNRFile, STDFile):
    plt.plot(data[1000:len(data)-1])
    plt.title("Original Measurement")
    plt.savefig(origFile)
    plt.close()
    data = SNR_Calculation_DM_ZeroSTD_Removal(data, Window_Size_SNR)
    plt.plot(data[1000:len(data)-1])
    plt.title("Sliding-Window SNR of Measurement")
    plt.savefig(SNRFile)
```

```

plt.close()
data = stddev(data, Window_Size_STD)
plt.plot(data[1000:len(data)-1])
plt.title("Sliding-Window Standard Deviation of SNR of Measurement")
plt.savefig(STDFile)
plt.close()
return data

# -----Main Program-----
# Set file parameters:
sourceFile = "test.csv"
VAM_col = 0
VBM_col = 2
VCM_col = 4
IAM_col = 8
IBM_col = 10
ICM_col = 12
# Set SNR processing parameters:
Window_Size_SNR = 200
Window_Size_STD = 200
orig_file_IAM = "output_validate/orig_IAM.png"
SNR_file_IAM = "output_validate/SNR_IAM.png"
STD_file_IAM = "output_validate/STD_IAM.png"
opfile_IAM = "output_validate/temp/IAM.csv"
orig_file_IBM = "output_validate/orig_IBM.png"
SNR_file_IBM = "output_validate/SNR_IBM.png"
STD_file_IBM = "output_validate/STD_IBM.png"
opfile_IBM = "output_validate/temp/IBM.csv"
orig_file_ICM = "output_validate/orig_ICM.png"
SNR_file_ICM = "output_validate/SNR_ICM.png"
STD_file_ICM = "output_validate/STD_ICM.png"
opfile_ICM = "output_validate/temp/ICM.csv"
orig_file_VAM = "output_validate/orig_VAM.png"
SNR_file_VAM = "output_validate/SNR_VAM.png"
STD_file_VAM = "output_validate/STD_VAM.png"
opfile_VAM = "output_validate/temp/VAM.csv"
orig_file_VBM = "output_validate/orig_VBM.png"
SNR_file_VBM = "output_validate/SNR_VBM.png"
STD_file_VBM = "output_validate/STD_VBM.png"
opfile_VBM = "output_validate/temp/VBM.csv"
orig_file_VCM = "output_validate/orig_VCM.png"
SNR_file_VCM = "output_validate/SNR_VCM.png"
STD_file_VCM = "output_validate/STD_VCM.png"
opfile_VCM = "output_validate/temp/VCM.csv"
#-----IAM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_IAM = original.iloc[:,IAM_col]
del original
gc.collect()
test_sample_IAM = np.asarray(test_sample_IAM)
print("IAM brought in!")
# Perform SNR analysis:

```

```

test_sample_IAM = exportSNRSTD(test_sample_IAM, Window_Size_SNR, Window_Size_STD, orig_file_IAM,
SNR_file_IAM, STD_file_IAM)
np.savetxt(opfile_IAM, test_sample_IAM, delimiter=',')
print("IAM SNR STD done!")
del test_sample_IAM
gc.collect()
#-----IBM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_IBM = original.iloc[:,IBM_col]
del original
gc.collect()
test_sample_IBM = np.asarray(test_sample_IBM)
print("IBM brought in!")
# Perform SNR analysis:
test_sample_IBM = exportSNRSTD(test_sample_IBM, Window_Size_SNR, Window_Size_STD, orig_file_IBM,
SNR_file_IBM, STD_file_IBM)
np.savetxt(opfile_IBM, test_sample_IBM, delimiter=',')
print("IBM SNR STD done!")
del test_sample_IBM
gc.collect()
#-----ICM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_ICM = original.iloc[:,ICM_col]
del original
gc.collect()
test_sample_ICM = np.asarray(test_sample_ICM)
print("ICM brought in!")
# Perform SNR analysis:
test_sample_ICM = exportSNRSTD(test_sample_ICM, Window_Size_SNR, Window_Size_STD, orig_file_ICM,
SNR_file_ICM, STD_file_ICM)
np.savetxt(opfile_ICM, test_sample_ICM, delimiter=',')
print("ICM SNR STD done!")
del test_sample_ICM
gc.collect()
#-----VAM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_VAM = original.iloc[:,VAM_col]
del original
gc.collect()
test_sample_VAM = np.asarray(test_sample_VAM)
print("VAM brought in!")
# Perform SNR analysis:
test_sample_VAM = exportSNRSTD(test_sample_VAM, Window_Size_SNR, Window_Size_STD,
orig_file_VAM, SNR_file_VAM, STD_file_VAM)
np.savetxt(opfile_VAM, test_sample_VAM, delimiter=',')
print("VAM SNR STD done!")
del test_sample_VAM
gc.collect()
#-----VBM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_VBM = original.iloc[:,VBM_col]
del original

```

```

gc.collect()
test_sample_VBM = np.asarray(test_sample_VBM)
print("VBM brought in!")
# Perform SNR analysis:
test_sample_VBM = exportSNRSTD(test_sample_VBM, Window_Size_SNR, Window_Size_STD,
orig_file_VBM, SNR_file_VBM, STD_file_VBM)
np.savetxt(opfile_VBM, test_sample_VBM, delimiter=',')
print("VBM SNR STD done!")
del test_sample_VBM
gc.collect()
#-----VCM-----
# Load in data of interest:
original = pd.read_csv(sourceFile, error_bad_lines=False)
test_sample_VCM = original.iloc[:,VCM_col]
del original
gc.collect()
test_sample_VCM = np.asarray(test_sample_VCM)
print("VCM brought in!")
# Perform SNR analysis:
test_sample_VCM = exportSNRSTD(test_sample_VCM, Window_Size_SNR, Window_Size_STD,
orig_file_VCM, SNR_file_VCM, STD_file_VCM)
np.savetxt(opfile_VCM, test_sample_VCM, delimiter=',')
print("VCM SNR STD done!")
del test_sample_VCM
gc.collect()
print("All data processed!")

```

Appendix B: SZS_validate.py

```
# This program preprocesses time-series data using the sum of Z-scores
# preprocessing technique, then performs support vector machines (SVM)
# classification of each sample in the data set (after downsampling).
# It trains itself on a fixed percentage (70% by default) of the data and
# performs a validation on the remaining percentage of the data.
# The validation results are printed out in a file with a row of predicted
# values and a row of actual values, identified according to a
# preclassification step.
# The number of correct and incorrect samples, along with the (trivially)
# calculated percent accuracy, are also printed to the standard output.
# Preclassification is done based on the number of samples until the fault.
# The time of the fault can be set in order to prevent classification of
# post-fault samples.
# After training and validation is complete, a file called "trained_model.svm"
# is exported with the trained SVM model included.
# The program "SZS_classify.py" can then be used to perform sample-by-sample
# classification of power system asset health on NEW DATA
# based on a previously trained model.
# By default, it obtains voltage and current SNR STD information in the folder
# called "output/temp".
# If you have not yet done so, please pre-process your synchrophasor CSV
# using "SNR_STD_exporter.py".
# Imported libraries:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import csv as csv
import pickle
import gc

# A function to standardize data to a standard-normal distribution
# (zero mean, unity standard deviation):
def standard_normal(data_to_be_scaled):
    # Use 'StandardScaler' from sklearn to standardize data:
    sc = StandardScaler()
    sc.fit(data_to_be_scaled)
    scaled_data = sc.transform(data_to_be_scaled)
    return scaled_data

# Support Vector Machine (SVM) (rbf) implementation function with export:
def ml_SVM_rbf_export(X_train_std, X_test_std, y_train, y_test, C_value, gamma_value, export_location):
    # Set up an SVM object with adjustable gamma and 'C' value:
    svm = SVC(kernel='rbf', random_state=0, gamma=gamma_value, C=C_value)
    # Fit the training data using the SVM model object:
    svm.fit(X_train_std, y_train)
    #Export trained SVM model:
    print("Exporting trained model for future use... ")
    with open(export_location, 'wb') as model_file:
        pickle.dump(svm, model_file)
    # Now use the trained SVM model to predict actual test data:
    y_pred = svm.predict(X_test_std)
    # Create combined-data (test and training) arrays for further testing:
```

```

X_combined_std = np.vstack((X_train_std, X_test_std))
# Perform prediction for overall data set:
y_combined_pred = svm.predict(X_combined_std)
# Return the predicted results, combined results:
return y_pred, y_combined_pred
def SVM_export(input_dataframe, C, gamma, export_location):
    # Split the dataframe into data and target arrays.
    X = input_dataframe.iloc[:, 1:input_dataframe.shape[1]].values
    y = input_dataframe.iloc[:, 0].values
    # Use sklearn's 'test_train_split' to split off 30% of the data for model testing:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
    # Standardize the training and test data to a standard-normal distribution:
    X_train_std = standard_normal(X_train)
    X_test_std = standard_normal(X_test)
    # Run SVM (rbf):
    y_pred, y_combined_pred = ml_SVM_rbf_export(X_train_std, X_test_std, y_train, y_test, C, gamma,
export_location)
    # Compute accuracy statistics:
    difference_vector = y_pred.astype(np.int64) - y_test.astype(np.int64);
    total_classifications = len(y_test)
    misclassifications = len(difference_vector[difference_vector != 0])
    return([y_pred, y_test, total_classifications, misclassifications])
# Sum of z-scores preprocessing function:
def szs(inp, Window_Size_SZS):
    # Get rid of NaNs:
    where_are_nans = np.isnan(inp)
    inp[where_are_nans] = 0
    # Compute the standard deviation and mean of the entire data set:
    mean = np.mean(inp)
    STD = np.std(inp)
    # Initialize storage arrays:
    sum_of_scores = 0
    outp = np.zeros(len(inp))
    # Determine sum for first 'Window_Size_SZS' worth of samples:
    for m in range(Window_Size_SZS - 1):
        # Add the Z-score:
        sum_of_scores = sum_of_scores + (inp[m] - mean)/STD
    # Determine sliding-window sum for each remaining sample in the series:
    for m in (range(len(inp) - Window_Size_SZS)):
        # Remove earliest Z-score from sum:
        sum_of_scores = sum_of_scores - (inp[m] - mean)/STD
        # Add latest Z-score to sum:
        sum_of_scores = sum_of_scores + (inp[m + Window_Size_SZS] - mean)/STD
        # Record latest limit count in output array:
        outp[m + Window_Size_SZS] = sum_of_scores
    return outp
# Sum of Z-scores plotter and exporter function.
# Sum of Z-scores series is returned.
def exportSZS(data, Window_Size_SZS, SZSFile):
    data = szs(data, Window_Size_SZS)
    plt.plot(data[10000:len(data)-1])
    plt.title("Sliding-Window Sum of Z-Scores")
    plt.savefig(SZSFile)
    plt.close()
    return data
# Downsampling function:

```

```

def downsample(data,ratio):
    ds_sum = 0
    k = 0
    output_index = 0
    ds_data = np.zeros(int(np.ceil(len(data)/ratio)))
    for index in range(len(data)):
        ds_sum = ds_sum + data[index]
        if k == ratio:
            ds_data[output_index] = ds_sum/ratio
            k = 0
            ds_sum = 0
            output_index = output_index + 1
        k = k + 1
    return(ds_data)
# Preclassifier function:
def classifier(data,alert_start_sample,alarm_start_sample):
    classifications = np.zeros(len(data))
    classifications[0:alert_start_sample] = 0
    classifications[alert_start_sample:alarm_start_sample] = 1
    classifications[alarm_start_sample:len(classifications)] = 2
    df = pd.DataFrame(data=np.transpose((classifications,data)))
    return(df)
# Main program (call functions to be run here):-----
# SVM parameters:
C = 2.5
gamma = 0.03
export_location = "trained_model.svm"
# Preclassifier sample number definitions:
# Sample number where the fault happens (only consider data before this):
restrict_from_end = 0
# Number of samples before the fault to consider samples to be "alert":
alert_start_sample = (restrict_from_end - 36000)
# Number of samples before the fault to consider samples to be "alarm":
alarm_start_sample = (restrict_from_end - 18000)
# Sum of Z-scores parameters:
#SZS_window_size = 324000
SZS_window_size = 3240
SZS_downsample_ratio = 15
# File locations:
STD_IAM_file = "output_validate/temp/IAM.csv"
STD_IBM_file = "output_validate/temp/IBM.csv"
STD_ICM_file = "output_validate/temp/ICM.csv"
STD_VAM_file = "output_validate/temp/VAM.csv"
STD_VBM_file = "output_validate/temp/VBM.csv"
STD_VCM_file = "output_validate/temp/VCM.csv"
SZS_dest_IAM = "output_validate/SZS_plot_IAM.png"
SZS_dest_IBM = "output_validate/SZS_plot_IBM.png"
SZS_dest_ICM = "output_validate/SZS_plot_ICM.png"
SZS_dest_VAM = "output_validate/SZS_plot_VAM.png"
SZS_dest_VBM = "output_validate/SZS_plot_VBM.png"
SZS_dest_VCM = "output_validate/SZS_plot_VCM.png"
SZS_SVM_results_file = "output_validate/SZS_SVM_results.csv"
#-----Sum of Z-scores-----
# Import data sets from files:
STD_IAM = np.loadtxt(STD_IAM_file, delimiter=",")
STD_IBM = np.loadtxt(STD_IBM_file, delimiter=",")

```

```

STD_ICM = np.loadtxt(STD_ICM_file, delimiter=",")
STD_VAM = np.loadtxt(STD_VAM_file, delimiter=",")
STD_VBM = np.loadtxt(STD_VBM_file, delimiter=",")
STD_VCM = np.loadtxt(STD_VCM_file, delimiter=",")
print("Data imported!")
# Restrict training and testing data to pre-fault samples
STD_IAM = STD_IAM[0:len(STD_IAM)-restrict_from_end]
STD_IBM = STD_IBM[0:len(STD_IBM)-restrict_from_end]
STD_ICM = STD_ICM[0:len(STD_ICM)-restrict_from_end]
STD_VAM = STD_VAM[0:len(STD_VAM)-restrict_from_end]
STD_VBM = STD_VBM[0:len(STD_VBM)-restrict_from_end]
STD_VCM = STD_VCM[0:len(STD_VCM)-restrict_from_end]
print("Data selected!")
# Perform sum of Z-scores:
SZS_IAM = exportSZS(STD_IAM, SZS_window_size, SZS_dest_IAM)
SZS_IBM = exportSZS(STD_IBM, SZS_window_size, SZS_dest_IBM)
SZS_ICM = exportSZS(STD_ICM, SZS_window_size, SZS_dest_ICM)
SZS_VAM = exportSZS(STD_VAM, SZS_window_size, SZS_dest_VAM)
SZS_VBM = exportSZS(STD_VBM, SZS_window_size, SZS_dest_VBM)
SZS_VCM = exportSZS(STD_VCM, SZS_window_size, SZS_dest_VCM)
del STD_IAM
del STD_IBM
del STD_ICM
del STD_VAM
del STD_VBM
del STD_VCM
gc.collect()
print("Sum of Z-Scores done!")
# Remove NaNs and infs for matrix profile:
SZS_IAM = SZS_IAM[np.logical_not(np.isnan(SZS_IAM))]
SZS_IBM = SZS_IBM[np.logical_not(np.isnan(SZS_IBM))]
SZS_ICM = SZS_ICM[np.logical_not(np.isnan(SZS_ICM))]
SZS_IAM = SZS_IAM[np.logical_not(np.isinf(SZS_IAM))]
SZS_IBM = SZS_IBM[np.logical_not(np.isinf(SZS_IBM))]
SZS_ICM = SZS_ICM[np.logical_not(np.isinf(SZS_ICM))]
gc.collect()
SZS_VAM = SZS_VAM[np.logical_not(np.isnan(SZS_VAM))]
SZS_VBM = SZS_VBM[np.logical_not(np.isnan(SZS_VBM))]
SZS_VCM = SZS_VCM[np.logical_not(np.isnan(SZS_VCM))]
SZS_VAM = SZS_VAM[np.logical_not(np.isinf(SZS_VAM))]
SZS_VBM = SZS_VBM[np.logical_not(np.isinf(SZS_VBM))]
SZS_VCM = SZS_VCM[np.logical_not(np.isinf(SZS_VCM))]
gc.collect()
# Downsample data:
SZS_IAM = downsample(SZS_IAM, SZS_downsample_ratio)
SZS_IBM = downsample(SZS_IBM, SZS_downsample_ratio)
SZS_ICM = downsample(SZS_ICM, SZS_downsample_ratio)
SZS_VAM = downsample(SZS_VAM, SZS_downsample_ratio)
SZS_VBM = downsample(SZS_VBM, SZS_downsample_ratio)
SZS_VCM = downsample(SZS_VCM, SZS_downsample_ratio)
print("Sum of Z-Scores downsampling done!")
gc.collect()
# Prepare SZS results for SVM classification:
df_IAM =
classifier(SZS_IAM,int(np.floor(alert_start_sample/SZS_downsample_ratio)),int(np.floor(alarm_start_sample/SZS_
downsample_ratio)))

```

```

df_IBM = pd.DataFrame(data=np.transpose(SZS_IBM))
df_ICM = pd.DataFrame(data=np.transpose(SZS_ICM))
df_VAM = pd.DataFrame(data=np.transpose(SZS_VAM))
df_VBM = pd.DataFrame(data=np.transpose(SZS_VBM))
df_VCM = pd.DataFrame(data=np.transpose(SZS_VCM))
del SZS_IAM
del SZS_IBM
del SZS_ICM
del SZS_VAM
del SZS_VBM
del SZS_VCM
gc.collect()
print("Sum of Z-Scores classification done!")
# Perform SVM testing on sum of Z-scores results:
df = pd.concat([df_IAM, df_IBM, df_ICM, df_VAM, df_VBM, df_VCM], axis=1)
del df_IAM
del df_IBM
del df_ICM
del df_VAM
del df_VBM
del df_VCM
gc.collect()
[y_pred, y_test, total_classifications, misclassifications] = SVM_export(df, C, gamma, export_location)
print("Total classifications: " + str(total_classifications))
print("Misclassifications: " + str(misclassifications))
print("Percent accuracy: " + str(100.0*(1-misclassifications/total_classifications)))
print(" ")
print("Sum of Z-Scores SVM validation done!")
# Export SVM classifier results:
with open(SZS_SVM_results_file, 'w', newline="") as file:
    writer = csv.writer(file)
    for i in range(len(y_pred)):
        writer.writerow([y_pred[i], y_test[i]])
print("Sum of Z-Scores SVM results exported!")
print(" ")
# Clear variables:
del df
del y_pred
del y_test
gc.collect()
print("Program done!")
print(" ")

```

Appendix C: SZS_classify.py

```
# This program preprocesses time-series data using the sum of Z-scores
# preprocessing technique, then performs support vector machines (SVM)
# classification of each sample in the data set (after downsampling).
# The SVM classifier results are printed out in a file and are also
# plotted and saved as a figure file.
# If you have not yet done so, please pre-process your synchrophasor CSV
# using "SNR_STD_exporter.py".
# Imported libraries:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
import csv as csv
import pickle
import gc
# A function to standardize data to a standard-normal distribution
# (zero mean, unity standard deviation):
def standard_normal(data_to_be_scaled):
    # Use 'StandardScaler' from sklearn to standardize data:
    sc = StandardScaler()
    sc.fit(data_to_be_scaled)
    scaled_data = sc.transform(data_to_be_scaled)
    return scaled_data
def SVM_import(X, C, gamma, export_location):
    # Standardize the training and test data to a standard-normal distribution:
    X = standard_normal(X)
    # Import the SVM model:
    file = open(export_location, 'rb')
    svm = pickle.load(file)
    # Now use the trained SVM model to predict actual test data:
    y = svm.predict(X)
    return y
# Sum of z-scores preprocessing function:
def szs(inp, Window_Size_SZS):
    # Get rid of NaNs:
    where_are_nans = np.isnan(inp)
    inp[where_are_nans] = 0
    # Compute the standard deviation and mean of the entire data set:
    mean = np.mean(inp)
    STD = np.std(inp)
    # Initialize storage arrays:
    sum_of_scores = 0
    outp = np.zeros(len(inp))
    # Determine sum for first 'Window_Size_SSZ' worth of samples:
    for m in range(Window_Size_SZS - 1):
        # Add the Z-score:
        sum_of_scores = sum_of_scores + (inp[m] - mean)/STD
    # Determine sliding-window sum for each remaining sample in the series:
    for m in (range(len(inp) - Window_Size_SZS)):
        # Remove earliest Z-score from sum:
        sum_of_scores = sum_of_scores - (inp[m] - mean)/STD
        # Add latest Z-score to sum:
        sum_of_scores = sum_of_scores + (inp[m + Window_Size_SZS] - mean)/STD
```

```

        # Record latest limit count in output array:
        outp[m + Window_Size_SZS] = sum_of_scores
    return outp
# Sum of Z-scores plotter and exporter function.
# Sum of Z-scores series is returned.
def exportSZS(data, Window_Size_SZS, SZSFile):
    data = szs(data, Window_Size_SZS)
    plt.plot(data[10000:len(data)-1])
    plt.title("Sliding-Window Sum of Z-Scores")
    plt.savefig(SZSFile)
    plt.close()
    return data
# Downsampling function:
def downsample(data,ratio):
    ds_sum = 0
    k = 0
    output_index = 0
    ds_data = np.zeros(int(np.ceil(len(data)/ratio)))
    for index in range(len(data)):
        ds_sum = ds_sum + data[index]
        if k == ratio:
            ds_data[output_index] = ds_sum/ratio
            k = 0
            ds_sum = 0
            output_index = output_index + 1
        k = k + 1
    return(ds_data)
# Preclassifier function:
def classifier(data,alert_start_sample,alarm_start_sample):
    classifications = np.zeros(len(data))
    classifications[0:alert_start_sample] = 0
    classifications[alert_start_sample:alarm_start_sample] = 1
    classifications[alarm_start_sample:len(classifications)] = 2
    df = pd.DataFrame(data=np.transpose((classifications,data)))
    return(df)
# Main program (call functions to be run here):-----
# SVM parameters:
C = 2.5
gamma = 0.03
export_location = "trained_model.svm"
# Preclassifier sample number definitions:

# Sample number where the fault happens (only consider data before this):
restrict_from_end = 0
# Sum of Z-scores parameters:
#SZS_window_size = 324000
SZS_window_size = 3240
SZS_downsample_ratio = 15
# File locations:
STD_IAM_file = "output_classify/temp/IAM.csv"
STD_IBM_file = "output_classify/temp/IBM.csv"
STD_ICM_file = "output_classify/temp/ICM.csv"
STD_VAM_file = "output_classify/temp/VAM.csv"
STD_VBM_file = "output_classify/temp/VBM.csv"
STD_VCM_file = "output_classify/temp/VCM.csv"
SZS_dest_IAM = "output_classify/SZS_plot_IAM.png"

```

```

SZS_dest_IBM = "output_classify/SZS_plot_IBM.png"
SZS_dest_ICM = "output_classify/SZS_plot_ICM.png"
SZS_dest_VAM = "output_classify/SZS_plot_VAM.png"
SZS_dest_VBM = "output_classify/SZS_plot_VBM.png"
SZS_dest_VCM = "output_classify/SZS_plot_VCM.png"
SZS_SVM_image_file = "output_classify/SZS_SVM_results.png"
SZS_SVM_results_file = "output_classify/SZS_SVM_results.csv"
#-----Sum of Z-scores-----
# Import data sets from files:
STD_IAM = np.loadtxt(STD_IAM_file, delimiter=",")
STD_IBM = np.loadtxt(STD_IBM_file, delimiter=",")
STD_ICM = np.loadtxt(STD_ICM_file, delimiter=",")
STD_VAM = np.loadtxt(STD_VAM_file, delimiter=",")
STD_VBM = np.loadtxt(STD_VBM_file, delimiter=",")
STD_VCM = np.loadtxt(STD_VCM_file, delimiter=",")
print("Data imported!")

# Restrict training and testing data to pre-fault samples
STD_IAM = STD_IAM[0:len(STD_IAM)-restrict_from_end]
STD_IBM = STD_IBM[0:len(STD_IBM)-restrict_from_end]
STD_ICM = STD_ICM[0:len(STD_ICM)-restrict_from_end]
STD_VAM = STD_VAM[0:len(STD_VAM)-restrict_from_end]
STD_VBM = STD_VBM[0:len(STD_VBM)-restrict_from_end]
STD_VCM = STD_VCM[0:len(STD_VCM)-restrict_from_end]
print("Data selected!")
# Perform sum of Z-scores:
SZS_IAM = exportSZS(STD_IAM, SZS_window_size, SZS_dest_IAM)
SZS_IBM = exportSZS(STD_IBM, SZS_window_size, SZS_dest_IBM)
SZS_ICM = exportSZS(STD_ICM, SZS_window_size, SZS_dest_ICM)
SZS_VAM = exportSZS(STD_VAM, SZS_window_size, SZS_dest_VAM)
SZS_VBM = exportSZS(STD_VBM, SZS_window_size, SZS_dest_VBM)
SZS_VCM = exportSZS(STD_VCM, SZS_window_size, SZS_dest_VCM)
del STD_IAM
del STD_IBM
del STD_ICM
del STD_VAM
del STD_VBM
del STD_VCM
gc.collect()
print("Sum of Z-Scores done!")
# Remove NaNs and infs for matrix profile:
SZS_IAM = SZS_IAM[np.logical_not(np.isnan(SZS_IAM))]
SZS_IBM = SZS_IBM[np.logical_not(np.isnan(SZS_IBM))]
SZS_ICM = SZS_ICM[np.logical_not(np.isnan(SZS_ICM))]
SZS_IAM = SZS_IAM[np.logical_not(np.isinf(SZS_IAM))]
SZS_IBM = SZS_IBM[np.logical_not(np.isinf(SZS_IBM))]
SZS_ICM = SZS_ICM[np.logical_not(np.isinf(SZS_ICM))]
gc.collect()

SZS_VAM = SZS_VAM[np.logical_not(np.isnan(SZS_VAM))]
SZS_VBM = SZS_VBM[np.logical_not(np.isnan(SZS_VBM))]
SZS_VCM = SZS_VCM[np.logical_not(np.isnan(SZS_VCM))]
SZS_VAM = SZS_VAM[np.logical_not(np.isinf(SZS_VAM))]
SZS_VBM = SZS_VBM[np.logical_not(np.isinf(SZS_VBM))]
SZS_VCM = SZS_VCM[np.logical_not(np.isinf(SZS_VCM))]

```

```

gc.collect()
# Downsample data:
SZS_IAM = downsample(SZS_IAM, SZS_downsample_ratio)
SZS_IBM = downsample(SZS_IBM, SZS_downsample_ratio)
SZS_ICM = downsample(SZS_ICM, SZS_downsample_ratio)
SZS_VAM = downsample(SZS_VAM, SZS_downsample_ratio)
SZS_VBM = downsample(SZS_VBM, SZS_downsample_ratio)
SZS_VCM = downsample(SZS_VCM, SZS_downsample_ratio)
print("Sum of Z-Scores downsampling done!")
gc.collect()
# Prepare SZS results for SVM classification:
df_IAM = pd.DataFrame(data=np.transpose(SZS_IAM))
df_IBM = pd.DataFrame(data=np.transpose(SZS_IBM))
df_ICM = pd.DataFrame(data=np.transpose(SZS_ICM))
df_VAM = pd.DataFrame(data=np.transpose(SZS_VAM))
df_VBM = pd.DataFrame(data=np.transpose(SZS_VBM))
df_VCM = pd.DataFrame(data=np.transpose(SZS_VCM))
del SZS_IAM
del SZS_IBM
del SZS_ICM
del SZS_VAM
del SZS_VBM
del SZS_VCM
gc.collect()
print("Sum of Z-Scores classification done!")

# Perform SVM testing on sum of Z-scores results:
df = pd.concat([df_IAM, df_IBM, df_ICM, df_VAM, df_VBM, df_VCM], axis=1)
del df_IAM
del df_IBM
del df_ICM
del df_VAM
del df_VBM
del df_VCM
gc.collect()
y = SVM_import(df, C, gamma, export_location)
# Plot results:
plt.plot(y)
plt.title("SVM Classification of Downsampled Data")
plt.savefig(SZS_SVM_image_file)
plt.close()
print("Sum of Z-Scores SVM classification done!")
# Export SVM classifier results:
with open(SZS_SVM_results_file, 'w', newline="") as file:
    writer = csv.writer(file)
    for i in range(len(y)):
        writer.writerow([y[i]])
print("Sum of Z-Scores SVM results exported!")
print(" ")
# Clear variables:
del df
gc.collect()
print("Program done!")
print(" ")

```

References

- [1] Transformer explodes into fireball at SRP substation in Avondale (AZFamily). [Online]. Available: <http://www.azfamily.com/story/32111291/transformer-explodes-into-fireball-at-srp-substation-in-avondale>
- [2] PG&E Charged with Crimes in 2019 California Wildfire (The New York Times). [Online]. Available: <https://www.nytimes.com/2021/04/06/business/energy-environment/pge-kincade-fire.html>
- [3] Z-Scores (University of West Georgia). [Online]. Available: https://www.westga.edu/academics/research/vrc/assets/docs/zScores_HANDOUT.pdf
- [4] Standard Score (Wikipedia). [Online]. Available: https://en.wikipedia.org/wiki/Standard_score
- [5] How is the SNR of a speech example defined? (University of California at Berkeley). [Online]. Available: <https://www1.icsi.berkeley.edu/Speech/faq/speechSNR.html>
- [6] S. Deepajothi, S. Selvarajan, "Performance evaluation of SVM – RBF kernel for classifying ECoG motor imagery", in *Int. J. Comp. Sci. and Telecom.*, vol. 4, no. 5, pp. 44-48, May 2013.
- [7] Intel® Xeon® E Processors (Intel Corporation). [Online]. Available: <https://www.intel.com/content/www/us/en/products/details/processors/xeon/e.html>
- [8] Ramponi, G., Protopapas, P., Brambilla, M., & Janssen, R. (2018). T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling. *ArXiv*, abs/1811.08295.
- [9] Alexander G., Dongyu L., Sarah A., Alfredo C. & Kalyan V. (2020). TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks. *ArXiv*, abs/ 2009.07769.
- [10] A. Abur and A. G. Expósito, *Power System State Estimation: Theory and Implementation*. CRC Press, 2004.
- [11] A. Monticelli, *State Estimation in Electric Power Systems: A Generalized Approach*, ser. *Power Electronics and Power Systems*. Springer, 1999.
- [12] F. C. Schweppe, J. Wildes, and D. P. Rom, "Power system static state estimation, Parts I, II, III," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-89, pp. 120–135, 1970.
- [13] A. Angioni, T. Schlsser, F. Ponci, and A. Monti, "Impact of pseudo-measurements from new power profiles on state estimation in low-voltage grids," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 1, pp. 70–77, Jan 2016.
- [14] E. Manitsas, R. Singh, B. C. Pal, and G. Strbac, "Distribution system state estimation using an artificial neural network approach for pseudo measurement modeling," *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 1888–1896, Nov 2012.
- [15] A. Bernieri, C. Liguori, and A. Losi, "Neural networks and pseudo-measurements for real-time monitoring of distribution systems," in *Proceedings of 1995 IEEE Instrumentation and Measurement Technology Conference - IMTC '95*, April 1995, pp. 112–119.
- [16] R. Singh, B. C. Pal, and R. A. Jabr, "Distribution system state estimation through gaussian mixture model of the load as pseudo-measurement," *IET Generation, Transmission Distribution*, vol. 4, no. 1, pp. 50–59, January 2010.

- [17] Y. Hu, A. Kuh, T. Yang, and A. Kavcic, "A belief propagation based power distribution system state estimator," *IEEE Computational Intelligence Magazine*, vol. 6, no. 3, pp. 36–46, Aug 2011.
- [18] P. Chavali and A. Nehorai, "Distributed power system state estimation using factor graphs," *IEEE Transactions on Signal Processing*, vol. 63, no. 11, pp. 2864–2876, June 2015.
- [19] L. Schenato, G. Barchi, D. Macii, R. Arghandeh, K. Poolla, and A. V. Meier, "Bayesian linear state estimation using smart meters and pmus measurements in distribution grids," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Nov 2014, pp. 572–577.
- [20] K. Emami, T. Fernando, H. H. Iu, H. Trinh, and K. P. Wong, "Particle filter approach to dynamic state estimation of generators in power systems," *IEEE Transactions on Power Systems*, vol. 30, no. 5, pp. 2665–2675, Sep 2015.
- [21] T. Nakagawa, Y. Hayashi, and S. Iwamoto, "Neural network application to state estimation computation," in *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Jul 1991, pp. 188–192.
- [22] P. Barbeiroa, H. Teixeira, J. Krstulovica, J. Pereiraa, and F. J. Soares, "Exploiting autoencoders for three-phase state estimation in unbalanced distributions grids," *Electric Power Systems Research*, 2015.
- [23] D. Singh, J.P.Pandey, and D. Chauhan, "Radial basis neural network state estimation of electric power networks," *IEEE International Conference on Electric Utility Deregulation, Restructuring and Power Technologies. Proceedings*, 2004.
- [24] A. Onwuachumba, Y.Wu, and M.Musavi, "Reduced model for power system state estimation using artificial neural networks," *IEEE Green Technologies Conference*, 2013.
- [25] A. Silverstein and J. Follum, "High-resolution, time-synchronized grid monitoring devices," *North American Synchrophasor Initiative*, Tech. Rep. NAPSI-2020-TR-004, March 2020, [ONLINE], available (2020/8/20) at <https://www.naspi.org/node/819>.
- [26] Y. Liu, "Beyond today's synchrophasor," in *NSF Workshop on Forging Connections between Machine Learning, Data Science, and Power Systems Research*, March 2020, [ONLINE], available (2020/8/20) at <https://sites.google.com/umn.edu/ml-ds4pes/presentations>.
- [27] X. Wang, Y. Liu, and L. Tong, "Adaptive subband compression for streaming of continuous point-on-wave and pmu data," *IEEE Transactions on Power Systems*, pp. 1–1, 2021.
- [28] H. M. Merrill and F. C. Schweppe, "Bad data suppression in power system static state estimation," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-90, no. 6, pp. 2718–2725, 1971.
- [29] D. M. Falcao, P. A. Cooke, and A. Brameller, "Power system tracking state estimation and bad data processing," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-101, no. 2, pp. 325–333, 1982.
- [30] A. Abur, A. Keyhani, and H. Bakhtiari, "Autoregressive filters for the identification and replacement of bad data in power system state estimation," *IEEE Transactions on Power Systems*, vol. 2, no. 3, pp. 552–558, Aug 1987.
- [31] H. Salehfar and R. Zhao, "A neural network pre-estimation filter for bad-data detection and identification in power system state estimation," *Electric Power Systems Research*, vol. 34, no. 2, pp. 127 – 134, 1995.

- [32] K. Mahapatra, N. R. Chaudhuri, R. G. Kavasseri, and S. M. Brahma, "Online analytical characterization of outliers in synchrophasor measurements: A singular value perturbation viewpoint," *IEEE Transactions on Power Systems*, vol. 33, no. 4, pp. 3863–3874, 2018.
- [33] M. N. Kurt, O. Ogundijo, C. Li, and X. Wang, "Online cyber-attack detection in smart grid: A reinforcement learning approach," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5174–5185, 2019.
- [34] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. Vincent Poor, "Smarter security in the smart grid," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, 2012, pp. 312–317.
- [35] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. V. Poor, "Machine learning methods for attack detection in the smart grid," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 8, pp. 1773–1786, 2016.
- [36] X. Niu, J. Li, J. Sun, and K. Tomsovic, "Dynamic detection of false data injection attack in smart grid using deep learning," in *2019 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2019, pp. 1–6.
- [37] Y. Chen, S. Huang, F. Liu, Z. Wang, and X. Sun, "Evaluation of reinforcement learning-based false data injection attack to automatic voltage control," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 2158–2169, 2019.
- [38] K. R. Mestav and L. Tong, "Learning the unobservable: High-resolution state estimation via deep learning," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2019, pp. 171–176.
- [39] —, "Universal data anomaly detection via inverse generative adversary network," *IEEE Signal Processing Letters*, vol. 27, pp. 511–515, 2020.
- [40] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. Vincent Poor, "Smarter security in the smart grid," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, 2012, pp. 312–317.
- [41] M. Zhou, Y. Wang, A. K. Srivastava, Y. Wu, and P. Banerjee, "Ensemble-based algorithm for synchrophasor data anomaly detection," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 2979–2988, 2019.
- [42] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pp. 582–588, 1999.
- [43] T. Schlegl, P. Seebck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-anogan: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, pp. 30 – 44, 2019.
- [44] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017.
- [45] O. Kosut, L. Jia, R. J. Thomas, and L. Tong, "Malicious data attacks on the smart grid," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 645–658, Dec 2011.
- [46] J. Kim, L. Tong, and R. J. Thomas, "Dynamic attacks on power systems economic dispatch," *48th Asilomar Conference on Signals, Systems and Computers*, November 2014.
- [47] L. Jia, J. Kim, R. J. Thomas, and L. Tong, "Impact of data quality on real-time locational marginal price," *IEEE Transactions on Power Systems*, vol. 29, no. 2, pp. 627–636, 2014.
- [48] J. Kim, L. Tong, and R. J. Thomas, "Subspace methods for data attack on state estimation: A data driven approach," *IEEE Transactions on Signal Processing*, vol. 63, no. 5, March 2015.

- [49] J. Liang, L. Sankar, and O. Kosut, "Vulnerability analysis and consequences of false data injection attack on power system state estimation," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3864–3872, 2016.
- [50] J. Zhang, Z. Chu, L. Sankar, and O. Kosut, "Can attackers with limited information exploit historical data to mount successful false data injection attacks on power systems?" *IEEE Transactions on Power Systems*, vol. 33, no. 5, pp. 4775–4786, 2018.
- [51] C. Jutten and J. Herault, "Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture," *Signal Processing*, vol. 24, no. 1, pp. 1–10, 1991.
- [52] P. Comon, "Independent Component Analysis, a new concept?" *Signal Processing*, vol. 36, pp. 287–314, Apr. 1994.
- [53] P. Brakel and Y. Bengio, "Learning Independent Features with Adversarial Nets for Nonlinear ICA," Oct. 2017, arXiv:1710.05050.
- [54] F. N. David, "Two combinatorial test of whether a sample has come from a given population," *Biometrika*, vol. 37, no. 1/2, pp. 97–110, 1950.
- [55] I. I. Viktorova and V. P. Chistyakov, "On the calculation of the power of the test of empty boxes," *Theory of Probability & Its Applications*, vol. 9, no. 4, pp. 648–653, 1964.
- [56] L. Paninski, "A coincidence-based test for uniformity given very sparsely sampled discrete data," *IEEE Transactions on Information Theory*, vol. 54, no. 10, pp. 4750–4755, Oct 2008.
- [57] P. K. Varshney, *Distributed Detection and Data Fusion*. Springer, 1997.
- [58] M. Pignati et al., "Real-time state estimation of the EPFL-campus medium-voltage grid by using pmus," in *2015 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb 2015, pp. 1–5.
- [59] F. Sossan, E. Namor, R. Cherkaoui, and M. Paolone, "Achieving the dispatchability of distribution feeders through prosumers data driven forecasting and model predictive control of electrochemical storage," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 4, p. 17621777, Oct 2016.
- [60] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid structural characteristics as validation criteria for synthetic networks," *IEEE Transactions on Power Systems*, vol. 32, no. 4, pp. 3258–3265, 2017.
- [61] K. R. Mestav, J. Luengo-Rozas, and L. Tong, "Bayesian state estimation for unobservable distribution systems via deep learning," *IEEE Transactions on Power Systems*, vol. 34, no. 6, pp. 4910–4920, 2019.
- [62] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233 – 268, 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1063520315001748>
- [63] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [64] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *International Conference for Learning Representations*, 2015.
- [65] M. G. Augusta and T. Kathirvalavakumar, "Pruning algorithms of neural networks — a comparative study," *Central European Journal of Computer Science*, vol. 3, no. 3, pp. 105–115, Sep 2013. [Online]. Available: <https://doi.org/10.2478/s13537-013-0109-x>
- [66] W. H. E. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, vol. 1, no. 1, pp. 7–24, Dec 1984. [Online]. Available: <https://doi.org/10.1007/BF01890115>

- [67] L. Wasserman, All of Statistics. Springer-Verlag New York, 2004.
- [68] ———, All of Nonparametric Statistics. Springer-Verlag New York, 2006.
- [69] M. J. Sanjari and H. B. Gooi, “Probabilistic forecast of pv power generation based on higher order markov chain,” IEEE Transactions on Power Systems, vol. 32, no. 4, pp. 2942–2952, July 2017.
- [70] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, “Matpower: Steady-state operations, planning, and analysis tools for power systems research and education,” IEEE Transactions on Power Systems, vol. 26, no. 1, pp. 12–19, Feb 2011.
- [71] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” CoRR, 2015. [Online]. Available: <http://arxiv.org/abs/1502.01852>
- [72] F. Chollet et al., “Keras,” <https://keras.io>, 2015.
- [73] R. Von Mises, “Über aufteilungs-und besetzungswahrscheinlichkeiten,” Revue de la Faculté des Sciences de l’Université d’Istanbul, vol. 4, p. 145163, 1939.
- [74] A. Hyvriinen and E. Oja, “Independent component analysis: algorithms and applications,” Neural Networks, vol. 13, no. 4, pp. 411 – 430, 2000.
- [75] I. Khemakhem, D. Kingma, R. Monti, and A. Hyvarinen, “Variational autoencoders and nonlinear ica: A unifying framework,” in Proc. 23rd Int. Conf. on Artificial Intelligence and Statistics, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2207–2217.
- [76] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>.
- [77] I. Roytelman and V. Ganesan, “Coordinated local and centralized control in distribution management systems,” IEEE Transactions on Power Delivery, vol. 15, no. 2, pp. 718–724, 2000.
- [78] P. Brady, C. Dai, and Y. Baghzouz, “Need to revise switched capacitor controls on feeders with distributed generation,” in IEEE PES Transmission and Distribution Conference and Exposition, vol. 2, 2003.
- [79] R. Walling, R. Saint, R. C. Dugan, J. Burke, and L. A. Kojovic, “Summary of distributed resources impact on power delivery systems,” IEEE Transactions on power delivery, vol. 23, no. 3, pp. 1636–1644, 2008.
- [80] Y. P. Agalgaonkar, B. C. Pal, and R. A. Jabr, “Distribution voltage control considering the impact of pv generation on tap changers and autonomous regulators,” IEEE Transactions on Power Systems, vol. 29, no. 1, pp. 182–192, 2013.
- [81] A. Woyte, V. Van Thong, R. Belmans, and J. Nijs, “Voltage fluctuations on distribution level introduced by photovoltaic systems,” IEEE Transactions on Energy Conversion, vol. 21, no. 1, pp. 202–209, 2006.
- [82] M. E. Baran, H. Hooshyar, Z. Shen, and A. Huang, “Accommodating high pv penetration on distribution feeders,” IEEE Transactions on Smart Grid, vol. 3, no. 2, pp. 1039–1046, 2012.
- [83] A. Bernstein and E. Dall’Anese, “Bi-level dynamic optimization with feedback,” in IEEE Global Conference on Signal and Information Processing, 2017.
- [84] X. Zhou, Z. Liu, C. Zhao, and L. Chen, “Accelerated voltage regulation in multi-phase distribution networks based on hierarchical distributed algorithm,” IEEE Transactions on Power Systems, vol. 35, no. 3, pp. 2047–2058, 2019.

- [85] P. Bagheri and W. Xu, "Model-free volt-var control based on measurement data analytics," *IEEE Transactions on Power Systems*, vol. 34, no. 2, pp. 1471–1482, 2019.
- [86] E. Pourjafari and M. Reformat, "A support vector regression based model predictive control for volt-var optimization of distribution systems," *IEEE Access*, vol. 7, pp. 93352–93363, 2019.
- [87] D. B. Arnold, M. Negrete-Pincetic, M. D. Sankur, D. M. Auslander, and D. S. Callaway, "Model-free optimal control of var resources in distribution systems: An extremum seeking approach," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3583–3593, 2016.
- [88] H. Nazaripouya, H. R. Pota, C. Chu, and R. Gadh, "Real-time model-free coordination of active and reactive powers of distributed energy resources to improve voltage regulation in distribution systems," *IEEE Transactions on Sustainable Energy*, vol. 11, no. 3, pp. 1483–1494, 2020.
- [89] R. R. Mohassel, A. Fung, F. Mohammadi, and K. Raahemifar, "A survey on advanced metering infrastructure," *International Journal of Electrical Power & Energy Systems*, vol. 63, pp. 473–484, 2014.
- [90] D. J. Marihart, "Communications technology guidelines for ems/scada systems," *IEEE Transactions on Power Delivery*, vol. 16, no. 2, pp. 181–188, 2001.
- [91] O. Vinyals, I. Babuschkin, D. Silver et al., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [92] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.
- [93] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998, p. 23.
- [94] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *International Conference on World Wide Web*, 2010, pp. 661–670.
- [95] A. Agarwal, D. Hsu, S. Kale, J. Langford, L. Li, and R. Schapire, "Taming the monster: A fast and simple algorithm for contextual bandits," in *International Conference on Machine Learning*, 2014, pp. 1638–1646.
- [96] B. D. Anderson et al., "Failures of adaptive control theory and their resolution," *Communications in Information & Systems*, vol. 5, no. 1, pp. 1–20, 2005.
- [97] K. J. Astrom and B. Wittenmark, "On self tuning regulators," *Automatica*, vol. 9, no. 2, pp. 185–199, 1973.
- [98] N. Christopeit and K. Helmes, "Strong consistency of least squares estimators in linear regression models," *The Annals of Statistics*, pp. 778–788, 1980.
- [99] H. Drygas, "Weak and strong consistency of the least squares estimators in regression models," *Zeitschrift fur Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 34, no. 2, pp. 119–127, 1976.
- [100] L. Ljung, "Consistency of the least-squares identification method," *IEEE Transactions on Automatic Control*, vol. 21, no. 5, pp. 779–781, 1976.
- [101] T. L. Lai, "Asymptotically efficient adaptive control in stochastic regression models," *Advances in Applied Mathematics*, vol. 7, no. 1, pp. 23–45, 1986.
- [102] R. C. Dugan and T. E. McDermott, "An open source platform for collaborating on smart grid research," in *IEEE Power and Energy Society General Meeting*, 2011.
- [103] W. Warwick, T. Hardy, M. Hoffman, and J. Homer, "Electricity distribution system baseline report," *Pacific Northwest National Laboratory, PNNL*, vol. 25178, 2016.

- [104] V. M. da Costa, N. Martins, and J. L. R. Pereira, "Developments in the newton raphson power flow formulation based on current injections," *IEEE Transactions on power systems*, vol. 14, no. 4, pp. 1320–1326, 1999.
- [105] A. P. Reiman, T. E. McDermott, M. Akcakaya, and G. F. Reed, "Electric power distribution system model simplification using segment substitution," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 2874–2881, 2018.
- [106] J. Yu, Y. Weng, and R. Rajagopal, "Mapping rule estimation for power flow analysis in distribution grids," *arXiv preprint arXiv:1702.07948*, 2017.
- [107] R. Rossius, G. Zenker, A. Ittner, and W. Dilger, "A short note about the application of polynomial kernels with fractional degree in support vector learning," in *European Conference on Machine Learning*. Springer, 1998.
- [108] T. L. Lai and C. Z. Wei, "Least squares estimates in stochastic regression models with applications to identification and control of dynamic systems,"
- [109] *The Annals of Statistics*, vol. 10, no. 1, pp. 154–166, 1982.
- [110] —, "Asymptotically efficient self-tuning regulators," *SIAM Journal on Control and Optimization*, vol. 25, no. 2, pp. 466–481, 1987.
- [111] S. Wang, J. Zhao, Z. Huang, and R. Diao, "Assessing gaussian assumption of pmu measurement error using field data," *IEEE Transactions on Power Delivery*, vol. 33, no. 6, pp. 3233–3236, 2017.
- [112] Y. Liao, Y. Weng, G. Liu, and R. Rajagopal, "Urban mv and lv distribution grid topology estimation via group lasso," *IEEE Transactions on Power Systems*, vol. 34, no. 1, pp. 12–27, 2019.
- [113] A. Hoke, R. Butler, J. Hambrick, and B. Kroposki, "Steady-state analysis of maximum photovoltaic penetration levels on typical distribution feeders," *IEEE Transactions on Sustainable Energy*, vol. 4, no. 2, pp. 350–357, 2013.
- [114] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965