Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain

by

Hafiza Syeda Zainab Kazmi

CIIT/FA17-RSE-013/ISB

MS Thesis

in

Software Engineering

COMSATS University Islamabad, Islamabad - Pakistan

Spring, 2019

COMSATS University Islamabad

Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain

A Thesis presented to

COMSATS University Islamabad

in partial fulfillment
of the requirement for the degree of

MS (Software Engineering)

by

Hafiza Syeda Zainab Kazmi
CIIT/FA17-RSE-013/ISB

Spring, 2019

Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain

A Post Graduate Thesis submitted to the Department of Computer Science as partial fulfillment of the requirement for the award of Degree of MS (Software Engineering).

Name	Registration Number
Hafiza Syeda Zainab Kazmi	CIIT/FA17-RSE-013/ISB

Supervisor:

Dr. Nadeem Javaid,
Associate Professor, Department of Computer Science,
COMSATS University Islamabad,
Islamabad, Pakistan.

Co-Supervisor:

Dr. Muhammad Imran,
Assistant Professor, Department of Computer Science,
College of Computer and Information Sciences,
King Saud University (KSU), Saudi Arabia.

Final Approval

This thesis titled

Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain

by

Hafiza Syeda Zainab Kazmi
CIIT/FA17-RSE-013/ISB

has been approved
for the COMSATS University Islamabad, Islamabad, Pakistan.

External Examiner:________________________________

Dr. Sharifullah Khan,
Associate Professor, Department of Computing,
School of Engineering and Computer Science (SEECS),
National University of Science and Technology (NUST), Islamabad, Pakistan.

Supervisor:______________________________________

Dr. Nadeem Javaid,
Associate Professor, Department of Computer Science,
COMSATS University Islamabad, Islamabad, Pakistan.

Co-Supervisor:____________________________________

Dr. Muhammad Imran,
Assistant Professor, Department of Computer Science,
College of Computer and Information Sciences,
King Saud University (KSU), Saudi Arabia.

HoD:____

Dr. Majid Iqbal Khan,
Associate Professor, Department of Computer Science,
COMSATS University Islamabad, Islamabad, Pakistan.

Declaration

I Hafiza Syeda Zainab Kazmi (Registration No. CIIT/FA17-RSE-013/ISB) hereby declare that I have produced the work presented in this thesis during the scheduled period of study. I also declare that I have not taken any material from any source except referred to wherever due that amount of plagiarism is within acceptable range. If a violation of HEC rules on research has occurred in this thesis, I shall be liable to punishable action under the plagiarism rules of the HEC and COMSATS University.

Date: July 20, 2019

Hafiza Syeda Zainab Kazmi
CIIT/FA17-RSE-013/ISB

Certificate

It is certified that Hafiza Syeda Zainab Kazmi (Registration No. CIIT/FA17-RSE-013/ISB) has carried out all the work related to this thesis under the supervision of Dr. Nadeem Javaid at the Department of Computer Science, COMSATS University Islamabad, Islamabad, Pakistan and the work fulfills the requirement for award of MS degree.

Date: July 20, 2019

Supervisor:

Dr. Nadeem Javaid,
Associate Professor,
Department of Computer Science.

Co-Supervisor:

Dr. Muhammad Imran,
Assistant Professor, Department of Computer Science,
College of Computer and Information Sciences.

HoD:

Dr. Majid Iqbal Khan,
Associate Professor,
Department of Computer Science.

DEDICATION

$\mathcal{D}$edicated

to my parents.

First and foremost, to my late mother, who prepared me to face the challenges with faith and patience. She was a source of inspiration towards my goals. At this moment, she is not here to support me but I feel her presence that encourages me to achieve my aims. May Allah (SWT) grant her Jannah (Ameen).

And

To my father, who always had confidence in me and gave me strength and encouragement throughout the life. He supported me during my hard times.

ACKNOWLEDGEMENT

All praise belong the creator of the world who gave me strength to do my work. I want to pay homage to my supervisor, Dr. Nadeem Javaid for the continuous support during my research, for his motivation, for his patience and enlightened supervision during my research. His valuable words will always serve me as a beacon of light throughout my life. I must express my profound gratitude to my family, friends and ComSens for their sincere support and continuous encouragement. I would like to thank to all of them who supported me throughout the whole period. This achievement would not have been possible without them.

ABSTRACT

Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain

Transmission rate is one of the contributing factors in the performance of Wireless Sensor Networks (WSNs). Congested network causes reduced network response time, queuing delay and more packet loss. To address the issue of congestion, we have proposed transmission rate control methods. To avoid the congestion, we have adjusted the transmission rate at current node based on its traffic loading information. Multi-classification is done to control the congestion using an effective data science technique namely Support Vector Machine (SVM). In order to get less miss classification error, Differential Evolution (DE) and Grey Wolf Optimization (GWO) algorithms are used to tune the SVM parameters. With an increase in the development of Internet of Things (IoTs), people have started using medical sensors for health monitoring purpose. The huge amount of health data generated by these sensors must be recorded and conveyed in a secure manner in order to take appropriate measures during critical conditions of patients. Additionally, privacy of the personal information of users must be preserved and the health records must be stored. IoT devices must be authorized for the eradication of counterfeited actions. The emerging blockchain is a distributed and transparent technology that provides an unalterable log of transactions. In this thesis, we have made a Remote Patient Monitoring (RPM) system using blockchain-based smart contracts which supports enrollments of patients and doctors in a health centre thereby increasing user participation. The enrollments' data of the participants is secured using blockchain. Our system monitors the patients at distant places and generates alerts in case of emergency. The sensitive health data is stored on a distributed IPFS storage. The prescription is provided by the doctors for the treatment of patients. The hospital anlyzes the doctors' reputation from the submitted reviews of patients. We have used smart contracts for authorization of IoT devices and provided a legalised and secure way of using medical sensors. Using the blockchain technology, forgery and privacy hack is reduced thereby increasing the trust of people in RPM. Furthermore, in the evaluation of first scenario, the comparative analysis has shown that the proposed approaches DE-SVM and GWO-SVM are more proficient than other classification techniques. DE-SVM and GWO-SVM have outperformed the benchmark technique GA-SVM by producing 3% and 1% percent less classification errors, respectively. For fault detection in WSN, we have induced four types of faults in the sensor readings and detected the faults using the

proposed Enhanced Random forest (ERF). We have made a comparative analysis with state of the art data science techniques based on two metrics i.e., Detection Accuracy (DA) and True Positive Rate (TPR). ERF has detected the faults with 81% percent accuracy and outperformed the other classifiers in fault detection. Similarly, for the second scenario, we have provided simulations that verify the successful deployment of smart contracts. The comparison is made based on cost and time. The contracts take considerably less amount of transaction and executon cost.

Journal publications

1 Kazmi, H.S.Z., Javaid, N., Awais, M., Tahir, M., Shim, S., Zikria, Y.B. **2019**. "Congestion Avoidance and Fault Detection in WSNs using Data Science Techniques". In Transactions on Emerging Telecommunications Technologies. ISSN: 2161-3915.

2 Zahid, M., Ahmed, F., Javaid, N., Abbasi, R.A, Kazmi, H.S.Z, Javaid, A., Bilal, M., Akbar, M., Ilahi, M. **2019**. "Electricity Price and Load Forecasting using Enhanced Convolutional Neural Network and Enhanced Support Vector Regression in Smart Grids." Electronics, 8(2), 122, EISSN 2079-9292.

Conference proceedings

1 Kazmi, H.S.Z., Nazeer, F., Mubarak, S., Hameed, S., Basharat, A., Javaid, N. **2019**. "Trusted Remote Patient Monitoring using Blockchain-based Smart Contracts.". In 14-th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA).

2 Kazmi, H.S.Z., Javaid, N., Imran, M., Outay, F. **2019** "Congestion Control in Wireless Sensor Networks based on Support Vector Machine, Grey Wolf Optimization and Differential Evolution.". In 11th Wireless Days Conference (WD).

TABLE OF CONTENTS

Dedication vi

Acknowledgements vii

Abstract viii

Journal Publications 140

Conference Proceedings 141

List of Figures xvi

List of Tables xvii

List of Abbreviations xviii

List of Symbols xx

1 Introduction 1

1.1 Introduction 2

1.1.1 Energy Efficiency 2

1.1.2 Trustfulness 4

1.1.3 Thesis Contributions 6

1.1.3.1 Data Science in WSN 6

1.1.3.2 Blockchain in IoT 6

1.1.4 Thesis Organization 7

1.1.5 Conclusion of the Chapter 7

2 Related Work and Problem Statement 8
2.1 Related Work 9
2.1.1 Data Science in WSN 9
2.1.2 Blockchain in IoT 13
2.1.2.1 Blockchain-based IoT 13
2.1.2.2 Blockchain in Healthcare 16
2.1.2.3 Critical Analysis 17
2.2 Motivation and Problem Statement 23
2.2.1 Subproblem 1: Data Science in WSN 23
2.2.2 Subproblem 2: Blockchain in IoT 23
2.3 Conclusion of the Chapter 26

3 Proposed Solutions for Congestion Avoidance and Fault Detection using Data Science in WSN 27
3.1 System Model: Data Science in WSN 28
3.1.1 Transmission Rate Adjustment Strategy 28
3.1.2 Congestion Avoidance in WSNs 29
3.1.3 Fault Detection in WSNs 33
3.1.3.1 Enhanced Random Forest 34
3.2 Conclusion of the Chapter 35

4 Proposed Solutions for Remote Patient Monitoring using Blokchain in IoT 36
4.1 Proposed Solutions: Blockchain in IoT 37
4.1.1 Enrollments 37
4.1.2 Patient Monitoring 38
4.1.2.1 IPFS Storage 39
4.1.3 Enterprise and Device Authorization 40
4.1.4 Rating and Reviews 41
4.2 Conclusion of the Chapter 41

5 Simulations and Results of Congestion Avoidance and Fault Detection using Data Science in WSN 42
5.1 Simulations and Results: Data Science in WSN 43
5.1.1 Original Dataset 43
5.1.2 Proficiency of proposed GWO-SVM and DE-SVM 43
5.1.3 Comparison of GWO-SVM and DE-SVM with other Classifiers 44
5.1.3.1 Genetic Algorithm based SVM 45
5.1.3.2 Random Forest 45
5.1.3.3 Naive Bayes 46
5.1.3.4 K-Nearest Neighbor 46
5.1.3.5 Stochastic Gradient Descent 47
5.1.3.6 Multilayer Perceptron 48
5.1.4 Prepared Dataset 50
5.1.4.1 Evaluation Measures 50
5.1.4.2 Data Learning Rate Results 51
5.2 Conclusion of the Chapter 56
6 Simulations and Results of Remote Patient Monitoring using Blokchain in IoT 57
6.1 Simulations and Results: Blockchain in IoT 58
6.1.1 Evaluation Metrics 58
6.1.1.1 Cost 58
6.1.1.2 Time 64
6.2 Conclusion of the Chapter 66
7 Conclusion and Future Work 67
7.1 Conclusion 68
7.1.1 Congestion Avoidance and Fault Detection using Data Science in WSN 68
7.1.2 Remote Patient Monitoring using Blokchain in IoT 68

7.1.3 Future Work 69
7.1.4 Congestion Avoidance and Fault Detection using Data Science in WSN 69
7.1.5 Remote Patient Monitoring using Blokchain in IoT 69

8 References 70

Appendices 78
.1 Detail of appendices 79
.2 Implementation of Proposed Solution 1: 80
.3 Implementation of Proposed Solution 2: 114

Journal Publications 140

Conference Proceedings 141

LIST OF FIGURES

3.1 Wireless Sensor Network . . . 29
3.2 System Model of Congestion Avoidance . . . 32
3.3 System Model of Fault Detection . . . 34

4.1 Overview of the Healthcare System . . . 38
4.2 Doctors Reviews and Ratings . . . 40

5.1 The Graph of M5P tree results . . . 46
5.2 Miss classification in k-NN . . . 47
5.3 Classification Errors Comparison . . . 49
5.4 Classification Errors Comparison . . . 49
5.5 Classification Errors Comparison . . . 49
5.6 Detection Accuracy of DE-SVM on all Faults . . . 51
5.7 Detection Accuracy of GWO-SVM on all Faults . . . 51
5.8 Detection Accuracy of GA-SVM on all Faults . . . 52
5.9 Detection Accuracy of k-NN on all Faults . . . 52
5.10 Detection Accuracy of NB on all Faults . . . 53
5.11 Detection Accuracy of ERF on all Faults . . . 53
5.12 Detection Accuracy of SGD on all Faults . . . 53
5.13 Detection Accuracy of MLP on all Faults . . . 54
5.14 Average True Positive Rate . . . 56

6.1 SCs Deployment . . . 59
6.2 Patient Monitoring Modular SCs Deployment . . . 60
6.3 Enrollments Functions Costs . . . 60
6.4 IoT Device Functions Costs . . . 61
6.5 Ethers (USD) in Deployment of SCs . . . 61
6.6 Rating and Review Functions . . . 62
6.7 Input Length and GAS Amount . . . 63
6.8 Input Length and Mining Time . . . 63
6.9 Average Execution Time of Encryption Techniques . . . 65

LIST OF TABLES

2.1 Summarized Related Work: Data Science in WSN 18
2.2 Summarized Related Work: Blockchain in IoT 20

5.1 Errors of all Classifiers 47
5.2 Fault Detection Accuracies of Classifiers at different Fault Rates . . 55
5.3 Average Accuracies of all Classifiers 56

6.1 Deployment Costs 64
6.2 Comparison between Traditional and Blockchain Systems 65

List of Abbreviations

R	Amount of Retransmission
AES	Advanced Encryption Standard
AI	Artificial Intelligence
ANN	Artificial Neural Network
DACs	Distributed Autonomous Corporations
DA	Detection Accuracy
DApp	Decentralised Application
DE	Differential Evolution
DES	Data Encryption Standard
DHT	Distributed Hash Table
D2D	Device-to-Device
ECOC	Error Correcting Code
EHR	Electronic Health Record
EOA	Externally Owned Account
ERF	Enhanced Random Forest
FP	False Positive
GaussianNB	Gaussian Naïve Bayes
GA	Genetic Algorithm
GWO	Grey Wolf Optimization
IDE	Integrated Development Environment
IoT	Internet of Things
IoV	Internet of Vehicles
IPFS	InterPlanetary File System
k-NN	K-Nearest Neighbor
LoRaWAN	Long Range Wide Area Network
LS-SVM	Least Square Support Vector Machine
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
MSE	Mean Square Error

MHLP	Multilevel Hybrid Protocol
NB	Naive Bayes
OC-SVM	One Class Support Vector Machine
OS-LS	Online Sparse Least Square
PHI	Protected Health Information
P2P	Peer to Peer
PoC	Proof of Collaboration
RF	Random Forest
RAE	Relative Absolute Error
RMSE	Root Mean Square Error
RASE	Root Relative Squared Error
RPM	Remote Patient Monitoring
SDN	Software Defined Networking
SIFF	Sibling Intractable Function Family
SC	Smart Contract
SVM	Support Vector Machine
SGD	Stochastic Gradient Descent
TA-LS	Trend Analysis Least Square
TPR	True Positive Rate
TP	True Positive
USD	United States Dollar
VN	Vehicular Network
WSN	Wireless Sensor Network

List of Symbols

ΔB	Buffer Occupancy Ratio
ΔC	Congestion Degree
β	Constant in Gain Fault
α	Constant in Offset Fault
$\acute{x}$	Faulty Reading
$\theta 1$	Lower Bound
x	Normal Reading
η	Noise
C	Penalty Ratio
ΔR	Transmission Rate
$\theta 2$	Upper Bound

Chapter 1

Introduction

1.1 Introduction

The network having hundreds of thousands of connected nodes through edges is known as complex network. Complex networks are scale-free networks and have nonlinear interactions. Furthermore, in this era of technology, a continued growth in the research of complex networks can be seen. Complex networks are increasing in popularity due to the increase in consideration of network structures [1]. Wireless Sensor Network (WSN) is considered to be a typical complex network [2]. As, the sensor nodes have limited energy and resources, they can communicate with the immediate peers only. For saving the energy consumption in a multi-hop transmission, complex WSNs need to have efficient data transmission mechanisms in order to enhance lifetime of the network. Similarly, the Internet of Things (IoTs) make a large scale-free network that requires energy efficient, secure and trusted transmission methodologies to send data over the complex networks.

1.1.1 Energy Efficiency

WSN consists of large number of scattered sensor nodes. The data sensed by the sensor nodes is sent to the sink or a base station. Sensor nodes are being used for: animal tracking, flood detection, forecasting of the weather data, monitoring of patients and vehicle monitoring. Sensor nodes independently perform some sensing task and carry out some processing. These sensor nodes communicate with each other in order to forward the collected data to sink node. Some nodes act as relay nodes. Relay nodes are used to collect the sensed data and route the data to the sink. WSNs are prone to communication failures. Sensor nodes have the ability to work in harsh environments. However, sensor nodes have a limited battery time, less memory and fast energy depletion [3].

Large number of sensors in a wide geographical area provide better transmission. Congestion at a node occurs if the arrival of data packets at a particular node are greater than the number of outgoing data packets. Congestion can cause packet loss and reduced response time. Response time of the network is described as the amount of time needed for a packet transmission from a sender to a receiver. Response time decelerates with the reduced network throughput in a congested network. Special considerations are required to deal with congestion in a WSN. The problem of congestion has been tackled by the adjustment of transmission rate at each node [4]. Several transmission rate control mechanisms have been proposed in the past years. The incoming and outgoing rate of data packets should be handled in order to avoid retransmission and packet loss. Congestion should be controlled at each hop to avoid the problem of packet loss [4]. Mechanisms to detect and avoid congestion can serve the purpose.

Data mining algorithms have been used to recognize such complex problems and make smart decisions. Learning techniques are categorized as supervised learning and unsupervised learning that works on labeled and non-labeled data, respectively. For the problem mentioned earlier, several classification methods are be used to classify the data and predict the right amount of transmission rate of sensor nodes in a WSN. Classification methods like Support Vector Machine (SVM), K-Nearest Neighbor (k-NN), Naive Bayes, Neural Networks (NN) and Decision Tree can be used to classify such data. The optimization algorithms like Genetic Algorithm (GA), Harmony Search Algorithm (HSA), Grey Wolf Optimization (GWO), Differential Evolution (DE), Firefly Algorithm (FA) and Particle Swarm Optimization (PSO) can be used to optimize the classifier parameters in order to accurately classify the data. Parameter can be optimized using nature inspired algorithms to reduce misclassification errors and to yellow increase the performance of WSNs.

WSNs are installed in unreceptive environments so that's why they are prone to failures. The type of failures can be software or hardware. The faults that occur can cause erroneous output during normal processes. For example, if a sensor gives inappropriate readings during a natural disaster, then, severe consequences might be faced due to the nonappearance of warnings. Stream of traffic in the sensor data may be increased due to the occurrence of faults and results in energy depletion of the sensor nodes. The energy of sensors should be saved so that they can carry out the processing efficiently during long periods that may range from days to years. For this, battery replacement is not a solution as most of the times, sensor nodes are deployed in unreachable sites. The data is collected at sensor level and the sensor nodes transmit it to the sink, however, the presence of faults in sensor readings may cause congestion in the network because of the ambiguous and false readings. Therefore, fault detection is an important factor to be catered in order to manage the sensor network efficiently and adequately [5].

The faults in the readings collected by the sensors can be divided in four types [6]; gain, offset, stuck-at and out of bounds fault. Gain fault is defined as the rate of change in the data sensed by the sensors. Offset fault refers to the inappropriate value being added to the sensed data. Stuck-at fault is defined as the zero variation in the data. Out of bounds occurs when the data lies out of the defined limits.

Briefly, in this thesis, we present two novel methods to avoid congestion at every hop in a WSN. The congestion problem is tackled by adjusting the transmission rate using SVM. The parameters of SVM are tuned using DE and GWO algorithms. We have injected the above mentioned four types of faults in the data set and applied fault detection techniques on the prepared datasets. The presented Enhanced Random Forest (ERF) is applied on the faulty data and a comparison is made with other classifiers to validate whether ERF outperforms the other classification methods or not. The fault detection accuracy is calculated using two metrics i.e., Detection Accuracy (DA) and True Positive Rate (TPR).

1.1.2 Trustfulness

In recent years, fast growing popularity and extensive development of IoT can be witnessed. IoT is being used in smart cities, smart cars, wearable devices, e-business and healthcare. Considerable increase in the number of medical patients has been observed in various countries. IoTand wearable devices have enhanced the patient monitoring quality and a large number of patients can be monitored remotely. Remote Patient Monitoring (RPM) allows the monitoring of patients outside the health centre thereby increasing the patient care and decreasing the appointments time and cost. The core functionality of RPM is the monitoring of patients through wearable devices and transmission of health readings for diagnosis and treatment. Healthcare devices are divided into three types e.g., stationary, embedded and wearable [7]. Stationary devices are those having physical location e.g., remote chemotherapy, embedded devices are implanted devices in a body e.g., deep brain stimulation and wearable devices are Body-worn e.g., insulin pump. As RPM is growing world-wide, concerns about secure transmission of Electronic Health Record (EHR) are increased. The sensitive health data can be accessed by unauthorized parties, so there is a motivation to secure the medical data transmission [8]. Centralized storage can be a single point of failure and it is risky to store the medical data on a centralized platform. The data stored on centralized database can be a target of hackers as it can be accessed and modified easily. For maintaining the integrity of medical data, a decentralized and distributed storage is needed. Blockchain seems to be a promising solution for tackling privacy and security concerns in RPM systems. Blockchain is a distributed technology that records the transactions and provides secure transparency by acting as a shared ledger. The nodes in a blockchain network can only be added after proper consensus. The events are recorded as an immutable log and can not be altered by any unauthorized party.

We have addressed the privacy and security issues in RPM using blockchain based Smart Contracts (SCs). For health monitoring, health readings collected using sensors are sent to the blockchain for analysis and timely health alert generation. SCs are used to maintain immutable log of the transactions being made in RPM. Automatic health notifications using blockchain increases reliance of patients in wearing medical sensors or devices. We have stored the EHR on a decentralized IPFS storage. Unique identifier named hash can be used to retrieve the associated content on InterPlanetary File System (IPFS). For security purpose, data hashes are encrypted using AES256. Consequently, blockchain storage constrained is tackled and privacy of EHR is preserved. After the health alerts generation, the prescription for the treatment is provided by the doctor. This prescription is rated and reviewed by the patient. The reviews and ratings are stored in blockchain for maintaining the immutability thereby increasing trust and quality in prescriptions provided by doctors. The personal information of patients and doctors is also recorded. Medical devices authorization is done by maintaining the possession history of device custodians or owners in SCs. In this way, the origin of devices can be tracked easily to avoid counterfeited actions.

Briefly in this work, we present two novel methods to avoid congestion at every hop in a WSN. The congestion problem is tackled by adjusting the transmission rate using SVM. The parameters of SVM are tuned using DE and GWO algorithms. We have injected the above mentioned four types of faults in the data set and applied fault detection techniques on the prepared datastes. The presented Enhanced Random Forest (ERF) is applied on the faulty data and a comparison is made with other classifiers to validate whether ERF outperforms the other classification methods or not. The fault detection accuracy is calculated using two metrics i.e., DA and TPR. Additionally, for RPM, smart contracts are written and tested. Encryption algorithms are compared in terms of execution time. The RPM system is evaluated based on cost and time.

Precisely, our proposed methods are energy efficient and provide better data transmission as proven using extensive simulations. The reduced miss-classification errors and a better fault detection ensures an efficient transmission in a WSN. Moreover, blockchain ensured secure transmission of data thereby increasing trust of the involved parties. The user prticipation and relaiability in IoT network is increased and the trustfulness is achieved.

1.1.3 Thesis Contributions

1.1.3.1 Data Science in WSN

This work is an extension of [10]. We have proposed two congestion avoidance techniques namely: Differential Evolution based Support Vector Machine (DE-SVM) and Grey Wolf Optimization-based Support Vector Machine (GWO-SVM). In this regard, we have taken the sensors readings from the dataset provided in [4]. Whereas, for fault detection, we have induced four types of faults: gain fault, offset fault, out of bound fault and stuck-at fault in the same dataset and prepared 20 new datasets having faulty readings. We have proposed Enhanced Random Forest (ERF) for the detection of faulty readings. Additionally, comparative analysis with other classification methods including: Genetic Algorithm based Support Vector Machine (GA-SVM), Random Forest (RF), Naive Bayes (NB), K-Nearest Neighbour (k-NN), Stochastic Gradient Descent (SGD) and Multilayer Perceptron (MLP) has also been performed to show the effectiveness of the proposed techniques.

1.1.3.2 Blockchain in IoT

This work is an extension of [11]. We have written the following blockchain-based SCs for healthcare system:

- Patients and Doctors Enrollments.
- Patients Health Monitoring.
- Modular SCs:
  1. Blood Pressure Monitor.
  2. Temperature Monitor.
  3. Blood Oxygen Monitor.
  4. Brain Inflammation Monitor.
- IPFS Storage.

- Enterprise.
- IoT Device Authorization.
- Rating and Reviews.

1.1.4 Thesis Organization

The rest of this thesis is organized as follows: Related work and problem statement are presented in Chapter 2. The proposed techniques for data science in WSN are presented in Chapter 3. The proposed solution for blolckchain in IoT is given in Chapter 4. Simulation results for data science in WSN are given in Chapter 5. Simulation results of blolckchain in IoT are given in Chapter 6 Finally, the conclusionand future work are presented in Chapter 7.

1.1.5 Conclusion of the Chapter

In this chapter, we have briefly discussed the introduction of data science techniques, optimization algorithms, fault detection and blockchain in WSNs. Additionally, the challenges faced by wireless sensor networks i.e., limited battery time, less memory and fast energy depletion are also discussed along with the proposed methodologies. Further, the existing literature about these challenges is elaborated in Chapter 2.

Chapter 2

Related Work and Problem Statement

2.1 Related Work

In this chapter, a review of the existing work in wireless sensor networks is presented. This chapter is divided in three sections. In section 2.1.1, existing work of data science in networks is presented. Section 2.1.2 covers the previous work done in netwroks blockchain. All the literature of data science andd blockchain in networks is summarized in tables 2.1 and 2.2. Finally, in section 2.2, the motivation for work and problem statements are discussed.

2.1.1 Data Science in WSN

The problem of congestion in wireless sensor networks is tackled in [4] by adjusting the transmission rate at current node. The node adjusts its transmission rate by taking buffer occupancy ratio and congestion degree estimate from the upstream node to avoid congestion and improve the network throughput. Multi-classification is done by SVM. The authors have used GA for parameter tuning. The parameters that are adjusted for all SVMs are acceptable error, penalty ratio and deviation of gaussian kernel function. Authors of [9] have proposed a clustering routing protocol in wireless sensor networks. The method used to enhance the performance and network lifetime is a three-level hybrid algorithm. The Multilevel Hybrid Protocol (MLHP) combined tree-based techniques. At level one, cluster heads are selected, whereas GWO is used for data transfer. To save energy, nodes select the best route using GWO. At level tree, distributed clustering is proposed. MLHP gives comparatively more residual energy, more stability and improved network lifetime in wireless sensor networks.

Finding location of unknown nodes is an important issue to be tackled. GWO [12] can be used for localization problems. Node localization problem articulates using range-based technique to calculate the coordinates of unknown nodes using the positions of the known nodes. the known nodes are called the anchor nodes which have a GPS device. Using the GPS device, the anchor nodes determine their positions. GWO gives better performance in terms of less computation time and success rate of localized nodes. This algorithm can be used in mobile networks as well. It can be combined with other heuristic algorithms for finding the location of nodes.

Deployment of sensor nodes in unreceptive environments causes the unreliable data collection. To gain the accurate information, anomaly detection mechanisms have been proposed earlier in research. In order to make decisions form the gathered data, it is noteworthy to detect anomalies in a sensor network. Discovering anomaly is an extensive process to determine its variance in behavior than the expected performance. Authors of [13] took the initiative to solve the one-class classification issue. The issue of anomaly detection is resolved by OCSVM. Support vector machine has been proven to be the efficient classification method. Radial base function can be used as kernel in OCSVM. Optimization of hyperparameters is used in OCSVM for anomaly detection.

Authors of [14] have catered the fault identification by initially classifying the sensor data using SVM. The sensor faults are detected using the proposed Online Sparse Least Squares Support Vector Machine (OS-LSSVM). The features of faults are extracted using Error Correcting Code Support Vector Machine (ECOC-SVM). The initial characteristics are separatedand the fault states are classified. ECOC-SVM and OS-LSSVM are considered to be highly efficient for real-time requirements of fault identification and prediction.

Sensor location is a key element that contributes in the performance of WSN because most of the applications in wireless sensor network domain need the known location of sensor nodes. Several optimization algorithms have been used to reduce the localization error of sensor nodes. Authors of [15] have used meta-heuristics to solve this optimization problem. Optimization algorithms like, PSO, FA and GWO algorithms are used to estimate the position of sensor nodes. The localization problem is resolved by minimizing the localization error using efficient optimization algorithms. GWO comparatively worked better and reduced more error than other algorithms. Node localization can be done by acquiring the information of anchor nodes. Anchor nodes are used to find the location of the target nodes. An optimization algorithm minimizes or maximizes the objective function. The location of sensors can be expressed or assessed as an objective in a meta heuristic algorithm to find an optimal solution. Grey wolf optimization technique works quite efficiently in this scenario. The grey wolves count is used to get the location of nodes and to minimize the error. Position of sensor nodes can be estimated using nature inspired algorithms like particle swarm optimization, firefly algorithm and many more. The authors found that GWO takes less computation time as compared to the other algorithms for localization problem. Distributed algorithms perform better as less transmission takes place to the base station and it helps in less energy depletion of sensors.

As anchor (nodes with known position) nodes are used to estimate the location of other sensor nodes, transmission range should be increased to localize more targets. However, it takes more computation time. As sensors have less energy and their energy depletes faster, providing a better network lifetime is challenging in WSNs. According to the authors of [16], GWO outperforms other optimization algorithms. It gives more accuracy and most importantly, GWO takes less execution time in an energy constrained environment.

Different type of failures can occur in WSNs and these failures can be categorized as software, hardware and communication failures. As, the sensors have limited resources, failures must be tackled. Detection should be efficient to avoid these failures. The best solution of failure detection in WSNs is machine learning. Faults in a WSN can occur at any time in a continuous manner or suddenly. To deal with this complexity, machine learning techniques can be of great help in the context of faults detection and recognition. For making decisions, classification methods seem to be the most appropriate solution. Authors of [6] have used SVM classification for detection of four types of faults such as gain fault, stuck-at fault, out-of-bound fault and offset fault. SVM has given satisfying results in detecting the faults in multidimensional data. The deployment in harsh environments causes the loss of data and the faulty data received from sensors can cause inadequate results. The detection determines the difference between non-faulty and faulty status.

The sensors are sparsely or densely deployed in wide hostile environments where they continuously share data. Quality of service is considered to be the most important factor for sufficient and accurate data communication. Appropriate fault detection techniques are required to deal with complex data. Authors of [5] have given a fault taxonomy and provided a survey of fault detection techniques. They have identified four types of soft faults i.e., offset, stuck-at, gain and out of bound. They have provided a comparison of algorithms for fault detection.

Almost every domain needs anomaly detection technique in order to avoid failures. [17] has given a OCSVM that works with streams. This technique is preferred because the data arrives at nodes is sequential in nature. To get a cost function, this online technique is formulated. The cost functions are minimized using SGD. They proved that the proposed technique detects faults accurately and gives higher true positive rate within less time and memory usage. Different types of faults like soft, hard and transient are known as heterogeneous. [18] has presented a protocol named as heterogenous fault diagnosis for sensors communication in WSNs. At first, it makes clusters, at second, it detects the faults and finally at third, it classifies the data taken from sensors to extract the faults. Hard faults are detected using time out strategy and the rest of the faults are detected using ANOVA test. The classification is done using feed forward Probabilistic Neural Network (PNN) to separate the faulty nodes in WSNs.

Many types of faults can occur in sensor networks that need to be handled according to their type. For this, recognition of fault is the most important factor. The state in which sensor is working should be monitored in order to detect faults. [19] has provided detection and identification scheme. The fault diagnosis scheme is used to detect and identify faults using Trend Analysis Least Squares Support Vector Machine (TA-LSSVM) and ECOC-SVM, respectively. At first, the failures or faults are detected by TA-LSSVM. At second, the ECOC-SVM differentiates various failures. The fault detection is done in a real time environment. Additionally, fault patterns are also identified using an improved algorithm.

The authors of [20, 21] have worked on increasing the localization accuracies and the results showed smaller localization error, higher localization accuracy however the presented techniques are time consuming and indoor environment is not considered. Machine learning methods are an effective way to classify sensors data [22, 23]. The errors generated due to misclassification are not acceptable in a WSN setting because it can cause harmful results. The authors of [24]-[28] have introduced the use of classification techniques for removal and reduction of errors. The authors of [29]-[34] have presented fault diagnosis and recovery schemes in order to effectively deal with the faulty readings. The authors of [35]-[39] have worked on reducing the amount of retransmissions and void holes and provided new ways of efficient routing in underwater WSN. The authors of [40] have used a depth threshold to the amount of hops and their presented metric ensures the packet delivery in underwater WSN. Authors of [41]-[44] have applied the machine learning techniques in WSN and concluded that network performance can be enhanced in terms of delays using these learning techniques. The authors of [45] have swarm intelligence for detecting and controlling the congestion in WSNs. The authors have claimed that old nature inspired routing enhances the network lifetime and reduces energy consumption. They have used firefly algorithm for a better data transmission by selecting the beets routing paths. Authors in [46] concluded that congestion causes a huge amount of energy wastage of sensors. They have worked on designing a better network topology in order to control congestion in a WSN.

In [47], a 5G network-based framework for vehicular clouds is proposed. Additionally, to control the congestion, a queuing strategy is implemented for a dense region i.e., for parking lots. Further, a resource allocation algorithm is proposed which enables the matching between the assigned tasks. As a result, candidate slices is developed. Simulation results validate that congestion is minimized at each access point (using control modules) of the network. The proposed scheme successfully enables the resource to job matching. However, the network has to face some delay.

Similarly, nowadays, a large amount of data is emitting from a number of connected devices. This data is centralized, resulting in delay. To address the aforementioned issue, researchers propose an idea of fog (by shifting the data processing and storage components nearer to the end-user). Later study validates that this solution is inefficient for spatial distribution. Therefore, in this work, Fog to Fog (F2F) collaboration is opted to mitigate the delay [48]. F2F promotes the offloading of incoming requests among the fog nodes and perform load balancing with resource management. Results show the effectiveness of the proposed scheme compared with other models. The summary of related work is given in Table 2.1.

2.1.2 Blockchain in IoT

The literature review of blockchin in networks is divided into two categories; IoT and healthcare. Section 2.1.2.1 explains the use of blockchain in IoT whereas section 2.1.2.2 gives an overview of the previous works done in healthcare using blockchain technology.

2.1.2.1 Blockchain-based IoT

Authors of [49] have provided an incentive mechanism for the protection of location information of users in a collaborative environment or a crowd sensing network. In the proposed system, the sensed data is sent to the confusion mechanism for protection against attacks. After going through the protection mechanism, the data is then sent to the blockchain that makes it tamper resistant and immutable. The experiment is done in a campus environment and the sensed data consists of user data, time, location and noise. Aim is to get more user participation. The encryption used in this mechanism proved to be more efficient than traditional non-encryption methods. The authors have concluded that males are more concerned about data protection as compared to females because more males have opted the presented protection mechanism. However, the results attained from a small sample are one-sided and the experiment scope is inadequate. Moreover, improved protection algorithms must be used in order to get a precise judgment.

Distributed Autonomous Corporations (DACs) [50] are the decentralized corporations that make decisions on their own without involving people. The authors have used DAC in IoT scenario by setting certain rules in order to automate the business industry. They have restructured the traditional model and used it in addition with smart property and paid data in a trade based blockchain. They have used bitcoins for trading and assured the transactions in DAC by SCs.

Edge computing origins enormous data sharing among different stakeholders. The parties feel resilient to share a huge amount of data with other untrusted parties. Therefore, it is required to gain trust among service providers and consumers. The authors of [51] have presented a big data sharing framework making use of the trusted blockchain. Resource constrained edges are supported using a less complex consensus mechanism PoC (Proof-of-Collaboration). The storage overhead is shortened using offloading and filtering schemes. They have made the use of hollow block and express transaction to enhance the communication efficiency.

Transmission of data in wireless networks is growing rapidly, however, it can be witnessed that wireless resources are inadequate [52]. Concurrent communication in the network causes interruption and this interruption can be avoided by temporarily disconnecting the users. The authors have validated the CSSI using blockchain. The presented consensus-based scheme is used to control the user access by cross-tier interference. The proposed method efficiently handles user access in mobile applications.

Big data is one of the important factors that needs to be catered by the research community [53]. The authors have presented a secure storage method for storing vehicle networking data using blockchain. Several nodes in a vehicular network are tackled using sub-blockchains. Data transmission in IoTs is effectively handled using distributed network and it provided a good storage mechanism.

Blockchain has ability to develop secure, intelligent, autonomous and more efficient transport systems. Infrastructure can be better utilized and transport system resources. Communication between vehicles is a developing trend that needs to be carefully handled. The authors of [54] presented a distributed blockchain based on vehicle network, Block-VN model. A network created between the vehicles for resource sharing and produces profitable services. Block-VN gives a distributed architecture for construction of a distributed transport management system.

It is difficult to manage revolutionary mobile communication and networking systems that are extremely complex. Authors of [55] presented AI as a solution to operate network autonomously. Mobile networks face data barriers due to their separation when they operate from distant places. In order to overcome these issues, a trustworthy data sharing framework is introduced. The model makes sure the tamper resistance state using blockchain. They used Hyperledger Fabric and SCs to control data access and provided a worthy data sharing environment.

Technical challenges are being faced in managing complex IoT networks. A huge number of frameworks have provided security features for the IoT devices, but the major problem is their centralized nature. It is hard to operate these centralized frameworks in IoT networks. Blockchain helps the purpose and gives a secure management of IoT devices due to its distributed nature. The solution proposed in [56] provides maximum scalability, less delays and more throughput as compared to other access management frameworks in IoT.

Many problems like security, flexibility and scalability have risen due to the diverse smart devices that make a vast IoT network. These issues are created because of the distributed nature of the IoT networks that can be tackled by implementing a centralized architecture. The authors of [57] have proposed a DistBlockNet model that combines the features of both centralized and distributed technologies: Software Defined Networking (SDN) and blockchain. It combines the benefits of SDN and blockchain and gives a more efficient architecture. Parties in IoT network can interact in a peer-to-peer manner by making use of blockchain. The authors have given a blockchain based technique to update a flow rule table that verifies and validates the table and enables the forwarding devices in IoT to get a latest flow rule table for effective communication.

The emerging Internet of Things (IoT) have smart devices connected that have the ability to generate and communicate data. This data can be of interest to the public for an evolving market of data trading where IoT devices owners trade their generated data to the users. Usually, monetization of IoT data involves intermediate parties that reduce the trust in trade. The authors of [58] have used SCs to automate the data trading and provided users with a trusted and decentralized platform.

Authors in [59] have tackled two challenges: lack of network coverage and the trust of network operators. Blockchain is implemented to minimize the threats and to achieve network security in sharing servers. The presented LoRaWAN provided a secure network when integrated with blockchain. However, linkage of various application servers with gateways needs a scaled architecture.

In the current era, service sharing among cloud servers and IoT devices is rapidly increasing that cause various security issues. To make service provisioning more secure and protected, blockchain is employed in [60]. Blockchain is used by the authors in network scenario to protect IoT devices or lightweight clients form insecure services. The authors have maintained validity states of services and edge servers in order to provide only legitimate and verified services to lightweight client via legal edge servers. The cloud servers and IoT clients have exchanged service codes and then IoT clients made inquiry to SCs about the validity state of the service in order to use a secure service. Consortium blockchain with proof of authority consensus mechanism is employed for secure services. However, edge server's authentication, charging of lightweight clients and service auditing need to be tackled by the authors.

A framework in [61] is presented for credibility verification of relationship among IoT and blockchain structures. A bubbles of trust named structure is provided by the authors of [62] for authentication of IoT devices thereby increasing data integrity. The authors of [63] have provided an access control mechanism for secure data sharing in smart grids and achieved more privacy. The blockchain-based model encourages customers to participate and increases profit generation. The authors of [64] also used blockchain for data trading of IoT devices and reduced various security risks by providing the confirmation of reputation of data using reviews. Similarly, data security issue is also tackled by [65] by using blockchain based access control strategy. The authors of [66] have provided a mechanism for service authentication using blockchain-based smart contracts. Transactions are made secure from malicious activities.

2.1.2.2 Blockchain in Healthcare

Authors of [67] tackled the privacy and security issues of EHR sharing using the immutable blockchain technology. Private and consortium blockchains are used for PHI sharing thereby increasing the privacy. The data is encrypted with keyword search. The proposed scheme achieved better data security and control over data access.

Medical research is increasing with an increase in medical accidents [68]. Healthcare is facing many threats like forgery, unauthorized access and record tracking.

The authors used provided verification of the proposed solution and concluded that the medical information is reliable and traceable using blockchain. Their data recovery function helps save the medical information against alteration.

The authors of [69] have proposed a framework to enable multiple users to collaborate and share documents in a trusted and secure manner. They have used blockchain for document sharing and version control exploiting the decentralized feature of blockchain. Their proposed solution eliminated the need of third party. IPFS is used for storage of documents. They have written ethereum SCs and tested the functionalities on Remix IDE. Research data reuse management using blockchain is done in [70]. Record of agreements among data owner and reuser are maintained as immutable log of blockchain.

Electronic Medical Records (EMRs) provide a way to store a huge amount of sensitive medical data yet it is difficult to share the personal data among health centres due to privacy concerns [71]. Blockchain provides a secure, trustworthy and tamper resistent maintenance of health records thereby enhancing data sharing. It is not feasible to store a huge amount of data on blockchain so, an IPFS storage is used to store the confidential data after masking. The solution provided data privacy due to data masking and the blockchain storage resources are saved using IPFS.

2.1.2.3 Critical Analysis

Many recent researches have explored efficient ways of using blockchain technology in various domains. However, the development still needs a rapid growth in order to exploit the distributed ledger in a more effective way. Paper [49] has provided a privacy protection mechanism, however further research can be done to explore blockchain privacy features. The authors of [50] have used blockchain for data trading, however the use of uniform data format is still needed. Papers [53] and [54] have used blockchain in vehicular networks for secure transport management. However, examination of proposed system from economic perspective needs to be catered in the presented scenario [53]. Proposed solution in [54] does not perform well as compared to optimized centralized IoT system in case of single hub. Distributed cloud computing architecture with secure fog nodes in IoT network needs further research [57]. Paper [60] proposes the use of blockcchain for secure service provisioning however, service auditing and charging of IoT devices must be taken under consideration in future research. Papers [67], [68] and [71] have used blockchain for RPM however, they have only stored the transactions in blockchain as blockchain has storage constraints. IPFS can be further studied to be used in future for the storage of sensitive medical data. The authors of [5]-[9] have proposed the use of blockchain technology and IPFS storage in healthcare and proved it to be a better privacy preserving and secure data storage mechanism for sensitive transactions. However, the use of IPFS storage can be risky as the data can be accessed by any party having the hash. The hash of data needs to be protected from unauthorized user. Future directions regarding the use of encryption techniques must be provided for a more secure storage.

Table 2.1: Summarized Related Work: Data Science in WSN

Objectives	**Optimization Algorithms**	**Classifiers**	**Achievements**	**Limitations**
Transmission rate adjustment [4]	GA	SVM	Improved network lifetime and throughput	No Fault Detection
Survey of fault detection techniques [5]	None	None	Shortcomings, advantages and future research directions for fault detection in WSNs	No fault prevention techniques are outlined
Fault detection and avoidance of negative alerts [6]	None	SVM	Classification of faults	No prediction of faults
Clustering routing protocol [9]	GWO	MHLP	Longer stability period, network lifetimeand more residual energy	No Hop-by-Hop Routing mechanism
Node localization [12]	GWO	None	Quick convergence and success rate	Only static network nodes are tested

Fault detection, identificationand isolation [13]	GA	Neural Network	Better detection accuracy, false alarm rate-and detection latency	Small stream of data is used for anomaly detection
Sensor Fault Detection and recognition of sensor failure [14]	None	SVM	Better identification accuracy, desirable real-time performanceand online fault identification	More energy consumption due to online identification
Reduction of localization error [15]	GWO, PSO and FA	None	Proficiency in determining the coordinates of nodes by minimizing the localization error	No centralized algorithms for localization are used
Sensor node localization [16]	GWO	None	Maximized network lifetime	Location of anchor nodes must be known
Anomaly detection [17]	None	OC-SVM	Better anomaly detection, higher true positive rate within less time and memory usage	Multiclass problems of classification are not taken into account
Heterogenous fault diagnosis [18]	None	Probabilistic Neural Network	Clustering, Fault Detection and Data Classification	No fault prevention and prediction

Fault detection and identification [19]	None	LS-SVM and ECOC-SVM	Improved real-time detection and identification accuracy	Manual parameter adjustment/ No optimization technique is used for parameter tuning
Localization in WSN [20]	GA	Neural Network	Improved localization	More time consuming and indoor environment is not considered
Acquire localization accuracy [21]	Linearization method, dynamic Weight PSO and DE	None	Smaller error of localization, higher localization accuracy and performance stability	Hybrid optimized method using DE and PSO algorithm can be used for better accuracy

TABLE 2.2: Summarized Related Work: Blockchain in IoT

Types of Networks	**Problems**	**Techniques**	**Contributions**	**Limitations/ Future Work**
IoT in crowd sensing [49]	Location privacy protection in crowd sensing networks	Blockchain based incentive mechanism	Protection of user's information using blockchain	Improved algorithm for best protection effect is needed

IoT in e-business [50]	An IoT E-business model that could fit for the E-business on the IoT	P2P trade based on the Blockchain	People can find required data on the platform and pay for the data provider	Uniform data format and API, ranking mechanism and credit system are needed
IoV [53]	Secure storage of vehicle networking data	Blockchains with different functions are designed	Integration of IoV with blockchain and analysis of transmission performance	Traffic among vehicles, channel reliability of cellular network
Vehicle network (VN) [54]	Security of drivers in vehicular environment	Block-VN distributed architecture	Privacy, Security and fault tolerance	Financing of mobile vehicles in a sharing economy
AI powered network [55]	Data barriers between diverse operators	Blockchain	Blockchain based data sharing framework for AI powered network	Examination of proposed system from economic perspective
Internet of things (IoT) [56]	Blockchain	Secure management of IoT devices	Maximum scalability, less delays and more throughput	Proposed solution does not perform well as compared to optimized centralized IoT system in case of single hub

IoT [57]	Diverse smart devices	Distblock net architecture	Secure SDN architecture for IoT using blockchain, updation of the flow rule tables	Building a distributed cloud computing architecture with secure fog nodes at the edge of the secure IoT network
IoT [58]	Intermediate parties reduce the trust in trade	Monetization of IoT data	Trust in data trading	DApps and wallets for different participants for interaction are needed
IoT [59]	The services IoT clients acquire are not secure	Secure network services for lightweight client using blockchain	Security and high throughput	Service auditing and charging of IoT devices is not taken under consideration
Sensor Network [67]	Privacy leakage of EHR	Blockchain	Blockchain based private EHR	No logs of the registered participants
Sensor Network [68]	No secure RPM	Blockchain-based RPM	Health alerts generation	No storage of health records
Sensor Network [69]	Storage of EMR	Blockchain	Storage on IPFS	No encryption
IoT [70]	IoT devices log maintenance	Blockchain	Tracking of device configuration changes	No authorization of devices

2.2 Motivation and Problem Statement

2.2.1 Subproblem 1: Data Science in WSN

Congestion is considered to be the most critical challenge as it affects the quality of service of sensor nodes. The main cause of congestion is the increased traffic condition in the wireless network where data transmission exceeds the occupation capacity of the sensor nodes. Substantial delays occur due to congested networks and large amount of packet loss takes place during data forwarding to the sink node. Authors of [4] have adjusted the transmission rate in WSN using SVM based on GA. In handling the congestion problem, SVM proved to be a better classification method. Authors of [6] have introduced faults in a multihop WSN and classified them as faulty and non-faulty readings taken from sensors using SVM. Better accuracy has been achieved using SVM in the presented scenario and [6] has ranked SVM as the best fault detection technique.

At first, to avoid the problem of congestion, a huge amount of packet re-transmissions, fast energy depletion and reduced throughput in WSN, a reliable transmission rate adjustment methodology is required. In [4], the authors have shown high classification error using GA and their proposed technique does not sufficiently reduce the amount of re-transmitted packets. Other optimization algorithms can be used to tune the SVM parameters that are capable of yielding better results than other classification methods. At second, detection of faulty data in WSN is an important issue to be catered as sensors are prone to errors because they are deployed in harsh environments. Extensive computations are not recommended for detection of faults because the sensor nodes have limited resources. Detection of faults should be made as precise as possible in order to extract the faulty status. The authors of [6] have used SVM to distinguish the faulty status of temperature readings recorded using sensors. We have used the same classification method SVM to avoid congestion. However, we have tuned the SVM parameters using DE and GWO algorithms. Additionally, anomaly detection in our scenario is done by ERF classifier.

2.2.2 Subproblem 2: Blockchain in IoT

The authors of [67] used blockchain for security and privacy-preservation of EHR. The immutability feature of blockchain helps in maintaining the tamper resistance state of records. The authors used private and consortium blockchains for tackling the privacy leakage issue of sensitive health data. Private blockchain is used to store the Protected Health Information (PHI) whereas, consortium blockchain maintains the indexes of the health record. For secure search of data, encryption with keyword search are used for patients' personal information and PHI.

The authors of [68] have proposed a model for sharing medical information exploiting the advantages of blockchain. They have used digital signatures for protection of medical information against forgery and unauthorized access. Medical information contains record number, date, time, doctor's ID, patient's name, patient's address, clinical health status, certificate ID and the digital signature of the record. They have verified the model to investigate wether it conforms to the security requirements for medical data sharing or not. The authors concluded that blockchain technology is reliable for medical data sharing.

Authors of [69] have presented a blockchain based data sharing scheme that facilitates the data sharing and changes tracking. It provides collaboration among multiple users to track changes on a document version without involving third party. The presented methods makes use of the ethereum SCs and a decentralized file system IPFS. IPFS is used for the storage of data. IPFS facilitates the user to track the version history of files stored thereby reducing the duplication and unauthorized access. The authors have provided a secure way of sharing digital content.

Researchers are reluctant to share their data due to protection concerns. A mechanism for stating terms of reuse of digital content is presented in [70]. The authors used blockchain and SCs for research data rights management. Externally Owned Accounts (EOA) for protection of data are used. The data publisher and data reuser use EOAs for an efficient and protected data sharing.

The demand for sharing IoT devices's generated data is increasing in enterprises. The authors of [71] have introduced a review system for the traded data in order to increase trust of data user on the enterprise. A review is given by the user to the enterprise. Blockchain is used to store the reviews submitted by the users and in this way, data reliability and reviews immutability is maintained.

Blockchain is suitable for the protection of confidential data. The authors of [72] used data masking for ensuring data privacy and implemented IPFS for a secure EHR. Data masking is applied on the patient's data e.g., name, age, ID, address and disease. IPFS storage is used to save blockchain resources because IPFS stores large files and the data can be retrieved using hash of the related content. However, they have used data masking instead of encryption. Data masking is applied using batch processes and the time taken to complete it is proportional to the size of data. As the data volume increases, data masking time will also increase.

Due to the dramatic increase in medical data gathered using medical devices, research community has adopted the use of IoT based wearable devices. Privacy concerns of patient's health data transfer have emerged due to these medical sensors. The authors of [7] have used blockchain technology for securing the data transactions log. They have employed a cloud for healthcare big data storage. The privacy and anonymity issues are tackled using cryptographic techniques and the IoT devices data is protected using the distributed blockchain technology. They have encrypted the data using cipher encryption and stored the data on cloud instead of blockchain. However, they have used the weak encryption technique and have not evaluated the performance of other symmetric encryption techniques for a fair comparison. The cloud storage can be a single point of failure. Also, they have not ensured the privacy of the patient's and medical professional's personal data. Their system does not maintain the logs of the registered and authorized participants in healthcare. Resultantly, the trust of patients in a health monitoring system is decreased.

The use of IoT devices based is increasing day by day thereby enhancing the comfort and lifestyles. The authors of [73] have suggested the use of blockchain technology for securing the IoT devices from tampering and unauthorized access. They have maintained the modifications being made in the device configurations utilizing the distributed features of blockchain. The modifications in configuration files are being stored in blockchain for improved security of IoT devices. However, they have used the hyperledger for implementation instead of using ethereum platform. Hyperledger uses no cyptocurrency and the transactions are confidential, not transparent. Moreover, they have not considered authorizing the enterprise who made the device (manufacturer) and device user's in order to avoid the counterfeited actions.

Remotely monitoring the patients helps in decreasing the cost thereby increasing the patient care outside the health centres. The increased number of IoT devices poses various privacy and security issues in a healthcare setting where confidentiality of patients' information must be maintained. The authors of [8] have used blockchain-based SCs for preserving the health data received from medical sensors. They have performed filtration of data gathered using sensors before writing it to the blockchain aiming to reduce the size and cost of data transactions. However, they have not maintained the profiles of patients and medical professionals that are enrolled in a health centre because of the privacy leakage issue and people will be unwilling to provide personal data. A decentralized and secure storage is needed because blockchain is not suitable for storage of huge amount of data. They have only stored the transaction logs of EHR and did not use any storage mechanism for EHR. Also, they have not done any authorization of the medical devices or sensors that raises the issue of counterfeited and fraudulent actions. A forged or fake device can be risky for a patient and the log of device authorization must be maintained without involving a third party. After the health alerts are generated, an online prescription should be sent to the patient for treatment of the disease. Additionally, patient must give review on the prescriptions given by the doctor. The publicly available reviews will increase other patients trust in RPM. Reviews can be maintained using the blockchain for the hospitals to analyze the reviews. The reputation of the doctors can be analyzed from this review system.

The problems we have identified include; personal information privacy concern, risky devices of patients, confidential EHR storage issue, no prescription for treatment, feedback of patients on treatment and ranking of doctor's prescriptions.

2.3 Conclusion of the Chapter

In this chapter, we have discussed the related work in detail about several problems of wireless sensor networks. We have described the limitations of existing work.The challenges are causing number of issues i.e., more energy consumption, cost maximization, more delay, congestion, errors generation, resource wastage, fault occurrence, forged sensors, etc. In Chapter 3, we have discussed the proposed methodologies in detail to maximize the congestion control, fault detection and sensors authorization which saves the energy consumption and cost. Additionally, the problem statements are also disscussed.

Chapter 3

Proposed Solutions for Congestion Avoidance and Fault Detection using Data Science in WSN

3.1 System Model: Data Science in WSN

The first subsection 3.1.1 explains the basic transmission rate strategy and second subsection 3.1.2 explains the proposed techniques for congestion avoidance. The third subsection 3.1.3 describes the second system model for fault detection in WSNs.

3.1.1 Transmission Rate Adjustment Strategy

Congestion occurs in WSNs when the number of incoming packets at a sensor node are more than the number of outgoing packets. 100 nodes were randomly deployed [4] in an area of 100m*100m with a sink and the congestion information was stored as shown in Fig. 1. The authors of [4] controlled congestion on each hop because bypassing the intermediate nodes in a WSN can not fully exclude the congestion. Two nodes are mainly considered to control the transmission rate i.e., downstream and current node. Transmission rate is increased or decreased based on the channel information of the downstream node. An awareness packet is sent from each node to the upstream node regarding the traffic information. Here, upstream node is the one from which the data is being received whereas, downstream node refers to the node that will receive the data. Normalized queue length ($\triangle B$) and congestion degree ($\triangle C$) are considered as traffic loading information. Based upon these two, traffic loading information is estimated at each sensor node. Normalized buffer size at any node v is defined as the ratio of number of packets in queue and buffer size. The buffer occupancy ratio of any node v can be calculated as [4]:

$$B_r(v) = \frac{Number\ of\ packets\ in\ queue\ buffer}{Buffer\ size\ at\ node\ v}. \tag{3.1}$$

Here, $B_r(v)$ represents the node traffic information that ranges from 0 to 1. A burst will occur if the node has low queue length and more packets enter queue of node v simultaneously. So, queue length metric is not fully adequate to recognise the data traffic at each node. Another metric named as congestion degree is used to analyse the changing tendency of buffer at a time period. Congestion degree [4] is calculated as the ratio of average processing time of packets and the interval of arrival time of two adjacent packets. The value is calculated as:

$$C_d = \frac{T_s}{T_a}. \tag{3.2}$$

If the current node has more traffic loading than the downstream node, then there is a need of increased transmission rate. Clearly, if the buffer occupancy of the current node is greater than the buffer occupancy of the downstream node, the current node is more congested and it should increase the data transmission rate. $\triangle B$ and $\triangle C$ determines the change in buffer occupancy ratio and congestion degree whereas, $\triangle R$ represents the increased or decreased data transmission rate. The amount of data transmission is determined on the basis of traffic loading information. In the transmission rate strategy, the transmission rate of the current node is determined by receiving the channel information of the downstream node. Normalized queue length and congestion degree are used to determine the transmission rate of each node. To determine the exact amount of data transmission, the amount of retransmission packets is deduced using different values of the traffic loading information. The packet loss or number of retransmission of packets is determined using the values of $\triangle B$, $\triangle C$ and $\triangle R$. Data transmission rate which gives the less amount of packet loss is selected. The abstract view of a WSN is shown in Fig. 3.1. It shows the overview of a WSN where sensors are deployed and the reading are being sent to the sink or base station. the readings are collected at sensor level and sent to the sink for further processing. The figure depicts the environment of a hop by hop transmission where congestion can occur at any node.

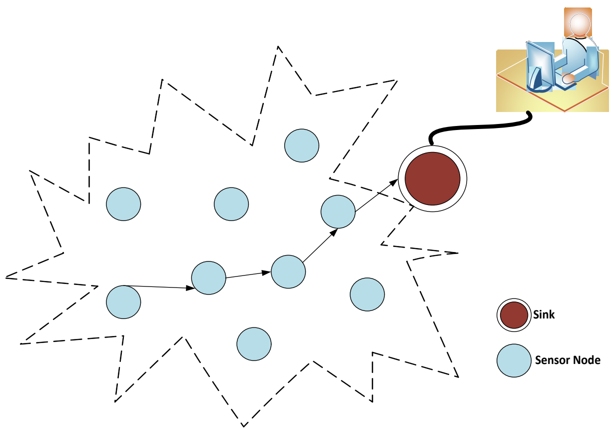

FIGURE 3.1: Wireless Sensor Network

3.1.2 Congestion Avoidance in WSNs

Sensor nodes must adjust the transmission rate according to the traffic loading information (normalized queue length and congestion degree) of each node in order to increase the network throughput. We have used the SVM classification method to ascertain the data transmission rate of each node in a WSN based on the transmission rate adjustment strategy explained in section [5]. Support Vector Machine: Suppose we have a data space X and we have to classify the data in two classes. We have d1, d2 . . . , dk data points or the training points with labels y1, y2 . . . , yk. we need to classify them in classes C or C1. The prediction is made whether the data point d belongs to a particular class or not. SVM can work efficiently on this problem. SVM [3] is used to separate the hyperplane optimally to classify the input data into positive or negative class. It produces the supreme distance between the data and the plane. A kernel function is used in non-linear classification to map the low-dimensional feature space classification data into a high-dimensional feature space. SVM [22] is a supervised learning machine that classifies the objects by finding a hyperplane. The hyperplane segments or divides the objects and determines in which category the object lies. Non-linear classification is done by changing the kernel function and generating hyperplane lines using Gaussian Radial Basis (RBF). SVM [23] uses different parameters like Penalty, Loss (loss function i.e., hinge and squared hinge), Dual (for optimization problem), Tol (for stopping criteria) and Random state (to generate random number). We have used the data set provided by [4] of 100 randomly deployed sensor nodes and used GWO and DE algorithms for SVM parameter tuning. The steps of the proposed work are taken as follows:

1. For sensor nodes, retransmission values are determined using the provided values of $\triangle B$, $\triangle C$ and $\triangle R$
2. The data is divided into independent variables and response variable that are $\triangle B$, $\triangle C$, $\triangle R$ and the number of retransmission packets, respectively. The $(\triangle B)$, $(\triangle C)$ and (ΔR) are used to interpret the retransmission values.
3. 80% and 20% data is used as training and testing data. SVM is designed for each retransmission value. Zero retransmission data values and other data values are labeled with 1 and -1, respectively. Five SVMs are designed for five retransmission values.
4. Grey Wolf Optimization Algorithm: Grey Wolf Optimizer is used to tune SVM parameters. Parameter tuning will help decrease the classification errors produced by the SVM. With more accurate classification, better transmission rate is selected. This transmission rate adjustment decreases packet loss and consequently increases congestion control in a WSN. The motivation of using GWO is taken from [15]. The adjusted parameters are penalty

ratio (C), acceptable error and the deviation of the gaussian kernel function. Maximum iterations and number of search agents taken are 50 and 5, respectively. GWO depicts the same mechanism as grey wolves hunting. Grey wolves always hunt in a pack. Each pack consists of four types of wolves that are alpha, beta, deltaand omega. Alpha wolves are known to be the leaders, the dominant members or more accurately the decision makers. Beta wolves support the alpha wolves and help them in decision making. Delta wolves follow the commands of alpha and beta. Omega are not considered an important entity. With a good hierarchy, each pack successfully hunts the prey. They track the prey, encircle and then harass it and attacks the prey when it attempts for self-defense.

- Social Hierarchy: Social hierarchy of grey wolves is distinguished into alpha, beta and delta which are considered as the best or optimum solution, second best and third best solution, respectively. Here the goal is to get a required solution or prey.
- Encircling Prey: It includes the encircling of a prey for an optimal solution. The values of A and C coefficient vectors can be adjusted in order to reach near the best agent.
- Hunting: The core of GWO algorithm is hunting. It means to move towards the solution and updating the alpha solution. With the alpha score, beta and delta can calculate their positions. The omega wolves are the remaining solutions and update themselves in reference with alpha, beta and delta solutions.
- Attacking Prey (exploitation): When the prey stops moving, the wolves attack the prey to finish the hunt. The fluctuation of the coefficient vector A is decreased by a. The random value A [-2a, 2a] where a is decreased from 2 to 0. With the operators, GWO search agents can update their positions using alphaand delta positions.
- Search for prey (exploration): Random population is generatedand the position of prey is estimated by alpha, beta and delta wolves. The distance of solution from prey is updated. To highlight exploration and exploitation, parameter a tends to decrease from 2 to 0.

5. Differential Evolution Algorithm: The motivation behind using the DE is taken from [46]. We have used DE to tune SVM parameters. The adjusted parameters are penalty ratio (C), acceptable error and the deviation of the gaussian kernel function. DE is an efficient algorithm that selects the optimal

solution from a random population. Therefore, we have used DE to select the best parameters for SVM. Suitable parameters help the classifier to classify the complex data accurately producing less classification errors. DE works the same way as GA. It performs crossover, mutation and selection. It takes two independent elements and accumulates the difference of these two.The they are multiplied by the mutation factor to generate a mutant element. The second step involves making the trial elements same as the population rate to perform crossover. The last step is known as selection as it selects the elements estimated in the previous step [24].

Figure 3.2 shows the system model of tuning of support vector machine parameters using grey wolf optimization and differential evolution algorithms. The datasets [4] are divided into train and test sets. In the train phase, GWO and DE are used to obtain the SVM parameters. The fitness value for each solution is estimated. The optimized parameters from GWO and DE are used to re-train the SVM. Then, the errors of classification are calculated which shows the amount of misclassification made by the proposed methods.

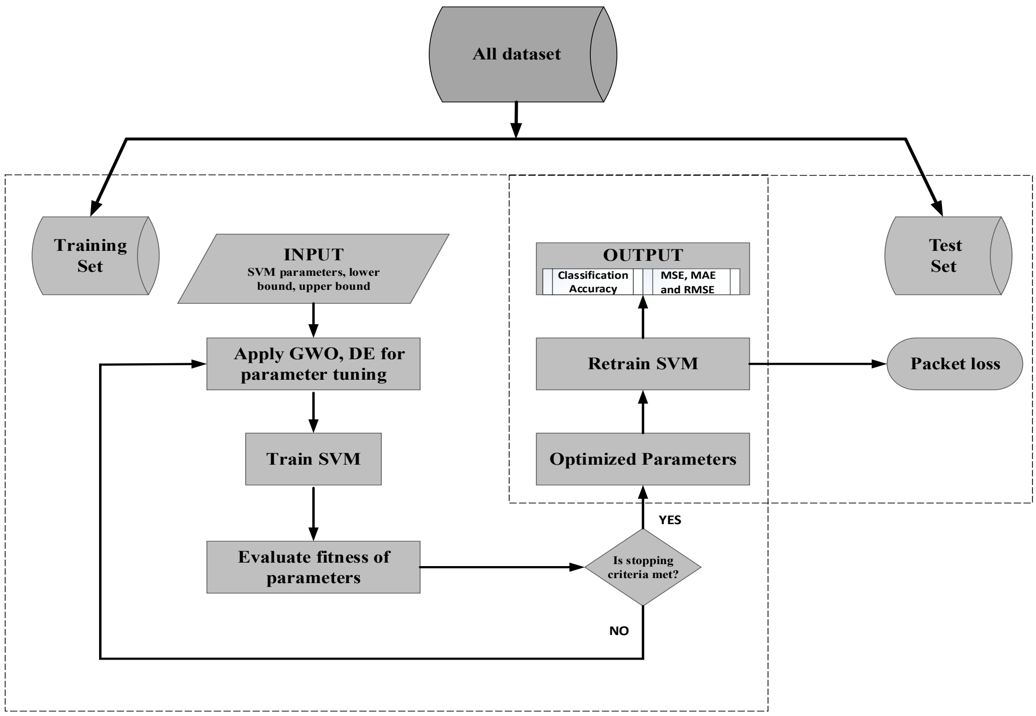

FIGURE 3.2: System Model of Congestion Avoidance

3.1.3 Fault Detection in WSNs

In the past years, WSNs have involved the research community in the advancement of wireless communication of sensors in a wide area. Consequently, the challenge arises to provide good quality of service results of failure and fault detection in a huge network. This challenge gives a motivation to proceed towards a fault taxonomy for WSNs and to provide fault detection techniques. As discussed earlier, the failures in WSNs are due to the faulty readings gathered by the sensors. The gathered data can be described as $d(n, t, f(t))$. Here, the node is represented by n and the sensed data by $f(t)$ at time t. Four types of faults are taken from [6] and are induced in the original sensor readings. The faults and equations [6] are described below:

1. Gain Fault: It is defined as the rate of change in the data sensed by the sensors. This error multiplies the sensed value by a constant and causes the change in rate of the sensed data due to poor calibration. Gain fault can be modeled as:
$$\acute{x} = \beta x + \eta. \tag{3.3}$$
where, x is the normal value sensed by the sensors, β is the constant multiplied and η represents the noise in the data. We have taken β and η as 4 and 0.8, respectively.

2. Offset fault: This fault refers to the inappropriate value being added to the sensed data. This type of faults occurs due to the inappropriate adjustment of the sensor. It causes a variation in the sensed data by addition of a value or constant.
$$\acute{x} = \alpha + x + \eta. \tag{3.4}$$
where, x is the normal value sensed by the sensors, α is the constant added to the normal reading and η represents the noise in the data. α is specified as 4 in our scenario.

3. Stuck-at Fault : It is defined as the zero variation in the data. Stuck-at fault occurs when the deviation in the sensed data collected by the node is zero. It can be defined as:
$$\acute{x} = \alpha. \tag{3.5}$$

4. Out of bounds: This fault occurs when the data lies out of the defined limits. Let [$\theta 1$,$\theta 2$] be the interval in which the sensed values lie, the out of bound

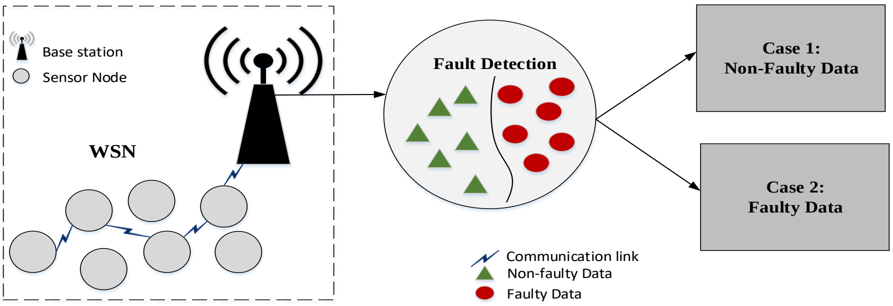

FIGURE 3.3: System Model of Fault Detection

fault can cause the readings to cross the threshold. It can be described as:

$$\acute{x} < \theta 1 \ or \ \acute{x} > \theta 2. \tag{3.6}$$

Figure 3.3 shows that the readings collected from sensors are sent to the sink and the readings are analyzed for faults. The readings are then detected and divided into faulty and non-faulty data. This faults identification enables us to have a better network traffic. For fault detection, we have proposed ERF that classifies the faulty and non faulty readings. ERF is explained in refERF

3.1.3.1 Enhanced Random Forest

The motivation of using and enhancing Random Forest (RF) is taken from [4]. ERFERF [22] includes different decision trees. Each decision tree analyses and votes on how the feature must be is classified. New items are classified based on voting done by the trees in the forests. We have applied random forest to classify the faulty and non faulty data. The accurate classification of the faulty readings help decrease the energy wastage in a WSN. We have enhanced the random forest classifier by selecting a less number of estimators for decision making. The parameters taken for are n_estimators = 5 and random_state = 42. The n-estimators represent the number of trees in a forest. It is proven that a large number of trees take much execution time, hence they do not provide an optimal solution in real time applications. ERF handles the categorical and missing values in a quite efficient manner. It can be used for feature ranking. It has the following hyper-parameters that can be adjusted in order to get accurate results i.e., criterion: to measure the split quality, max_features; maximum features, max_depth; height of the tree and min_samples_split; samples required before splitting. If the number of samples is greater than a threshold value then the node is split, min_samples_leaf; minimum number of data points allowed in a leaf node.

The data science techniques provide the advantage of data classification over the other techniques for congestion control. As per the literature survey, the multi classification feature of SVM gives overwhelming results during the data classification and its feasible for congestion estimation. Also, the data science technique i.e., SVM provides more accurate results when implemented for transmission rate adjustment. SVM classification method helps in congestion avoidance thereby increasing the efficiency of the network. On the implementation side, all the distributed systems work on the principle of peer-to-peer communication and need traffic balancing. By using the SVM classification method, hop-by-hop communication is made possible where power consumption in the data fusion process can be reduced. Possibly better results can be formulated using this effective data science technique. The best-suited parameters of SVM help in increased accuracy of the results. Optimization algorithms like GA, DE, GWO are ranked best for parameters selection. Briefly, the data science techniques are best to estimate the congestion at every hop in the network rather than controlling the congestion only at the sink node. The data science techniques used for classification provide better congestion-free WSN. Therefore, firstly, we have proposed SVM based techniques to avoid congestion in WSN (occurs due to increased transmission rate). This increased congestion results in packet loss, throughput reduction and low energy efficiency, which affects the quality of services. We have used the enhanced version of RF classifier for fault detection and it proved to be the best classification technique for the detection of faulty readings in a distributed setting. So, where ever the issues of congestion and fault detection will occur, our schemes will efficiently deal with the sensor readings and will provide an efficient transmission.

3.2 Conclusion of the Chapter

In this chapter, we have explained the proposed techniques in detail. Additionally, pseudocodes of the tehniques are also presented for better understanding of work done. In Chapter 5, the simulation results of the proposed solutions are discussed and the techniques are validated and discussed. The thesis is then concluded in Chapter 7.

Chapter 4

Proposed Solutions for Remote Patient Monitoring using Blokchain in IoT

4.1 Proposed Solutions: Blockchain in IoT

The subsection 4.1.1 explains the enrolments smart contract. Subsection 4.1.2 explains the remote patient monitoring contract, subsection 4.1.2.1 describes the storage used to store the medical records. Section 4.1.3 explains the IoT device authorizatiion contract and 4.1.4 describes the review sytem used in RPM.

In our scenario, medical sensors are embodied on patient's body and the health readings are sent to the specific SC via an master device i.e., a smart phone. The user interface on the master device is in charge for communication among blockchain and user. The patient profiles are managed by health centre using SCs. Patient's health status is analyzed according to the data being received. Health data is stored on a decentralized IPFS storage. Patients and doctors are able to register or enrol themselves using the master device. The health centre is in charge to authorize a patient for a doctor. Additionally, IoT device possession details are also recorded in SCs. Whenever an enterprise manufactures a device, SCs are made by both the enterprise and the patient who takes possession of the device. The main SCs named patient monitoring, enrolments, enterprise, IoT device authorization and IPFS storage for EHR are discussed below in detail.

4.1.1 Enrollments

Health centre initializes enrolments SC on the blockchain for the initiation of the doctors and patients' registrations. The enrolments contract consists of enrolment, modification and authorization functions. As shown in figure 4.1, health centre entity generates a public and a private key. Then, it posts the SCs address on the smart phone for patients and doctors to get registered easily in a secure way. The patient and doctors register in a health centre using their own EOAs via SCs address using *addpatient()* and *adddoc()* functions. The information taken from patient and doctors includes id, name, address and age and is made secure using EOA due to privacy concerns. Personal information is made private so that patient and medical assistants do not suffer from confidential information theft. In this way, patients and doctors will not be reluctant to enroll themselves due to the fear of privacy leakage and participation in the health system will be increased. The enrolments contract also allows the modification of information of both patients and doctors using *modifypatient()* and *modifydoc()* functions. Also, only a specific doctor is allowed to check the health status of a patient.

The health centre maintains a list of doctors and can authorize and deauthorize a doctor from monitoring a patient's health. The is done using *authorize()* and *deauthorize()* functions. Patients can view their information and authorized doctors by means of EOA. The enrolment, information modification and patient authorization details can be seen anytime.

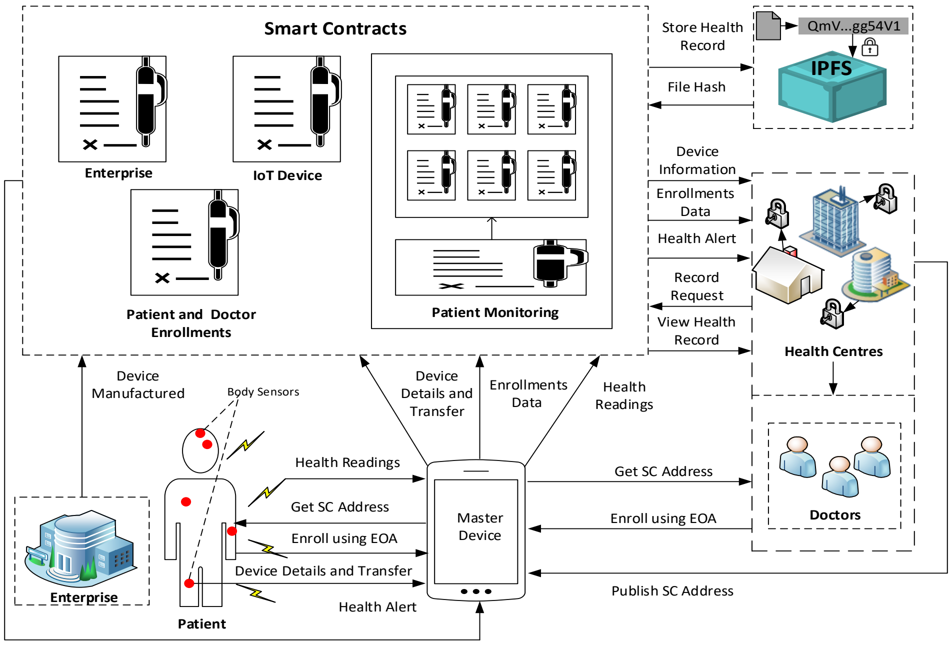

FIGURE 4.1: Overview of the Healthcare System

4.1.2 Patient Monitoring

For patients' monitoring, data received from the smart device is handled by the main SC named as HealthContractCaller. Then, the main HealthContractCaller contract creates a specific contract for every individual device it is getting data from. The main contract is like a container that organizes and creates links among all devices and relevant subcontracts for patient monitoring as shown in Fig 4.1. Authorized doctors or doctors are allowed to access patients' information and will be able to change thresholds for monitoring purpose.

For instance, if the smart device receives blood pressure data from a patient's body sensor, the data will be sent to Health Contract Caller and subsequently, *BloodPressureMonitor()* function will be called for patient monitoring. Minimum and maximum blood pressure values will be sent by the device to this function and an object is created by this function. Then, the individual subcontract Blood Pressure Monitor will pass these values to its analyze() function in order to evaluate the received data. If the *analyze()* function returns any value other than zero (0) or “OK”, then an alert (e.g. high/low blood pressure) is sent to the patient, doctor and health centre for treatment. The subcontracts we have used to monitor patient status include; Heart Rate Monitor, Glucose Monitor, Blood Pressure Monitor, Temprature Monitor, Blood Oxygen Monitor and Brain Inflammation Monitor. The motivation of modular contracts i.e. Heart Rate Monitor and Blood Sugar Level is taken from [8]. Whereas, we have proposed the other four subcontracts. The subcontracts analyse the real time heart rate, sugar level, fever, oxygen level in blood and brain inflammation based on specific threshold values. The modular contracts provide uncomplicated, trouble-free and simple maintenance. These modules will allow a customized structure where any subcontract for a specific device can be changed without changing the functionality of others.

In our system, blockchain is used to store the transactions only because blockchain is not best suited for storing a huge amount of data. We have used IPFS for storage of health data. The combination of IPFS and blockchain is powerful as it allows the IPFS hash to be stored on blockchain and the sensitive data on IPFS can be retrieved anytime using the hash links. The storage procedure is as follows:

4.1.2.1 IPFS Storage

The health data is sent to the IPFS for storage. The data files are awarded with a unique identifier known as cryptographic hash. The hash of the data is generated which will be used to fetch the content associated with it. IPFS searches the nodes and gets the required file. Storing the original hash of file is risky because any party having that hash can retrieve the file from IPFS. So, the IPFS hashes are encrypted using AES256 encryption technique for securing the data hash from any unauthorized party. In IPFS, version histories of files can be tracked thereby reducing the problem of duplication. Additionally, the authenticity of confidential data is achieved through this mechanism because the stored data can be found on the network nodes behind the unique identifiers.

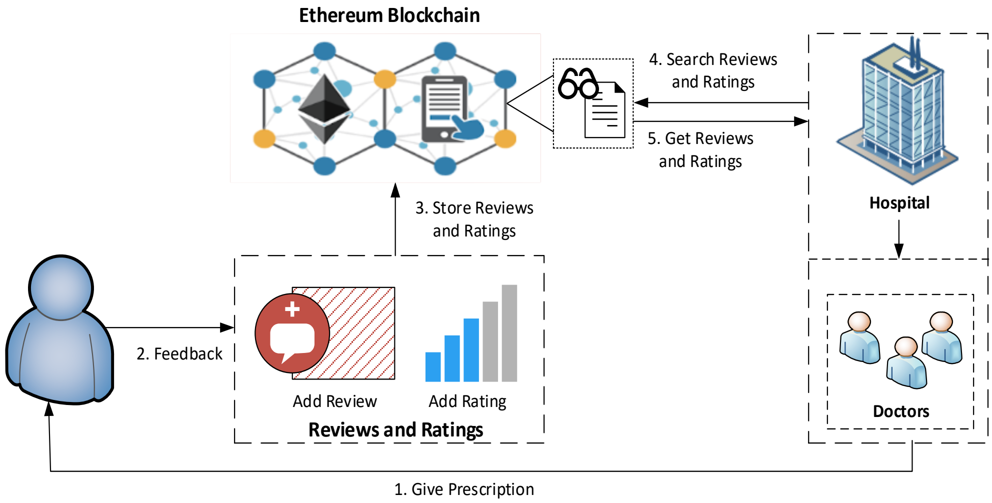

FIGURE 4.2: Doctors Reviews and Ratings

4.1.3 Enterprise and Device Authorization

There are two types of SCs for device authorization, one is of the enterprise and other is of the device custodian. Here, IoT device refers to the wearable body sensor of the patient. The patient having that IoT device is referred as custodian of the device. Device must be registered and the custody must be recognised. The patient who buys a device must get registered and the device credentials must be legalised. In traditional systems, the contracts were made by involving a third party. However, third parties are run by people that can be deceitful. We have established device credential management by removing the third party through SCs. The enterprise who manufactured the device initiates a SC named *newdevice()* after the production of device as shown in figure 4.1. Whenever a patient buys that medical device, it must make a contract to get registered as the custodian of device. The device custodian also initiates a SCs and stores device information like device name and device description. In this way, device management will be done by the patient. The device custodian can set access conditions and transfer the device possession to other parties in a decentralized manner. The transfer of possession function changes the possession using the current (registered) and new custodian (to be registered) addresses and changes the credentials of the device. The updated IoT device and custody details will also be sent to the health centre.

4.1.4 Rating and Reviews

After the health alerts generation, prescription for the treatment of disease is provided by the doctor to the patient. The patient gives reviews and ratings on the prescription whether it is effective or not. The rating and review SC is initiated by the hospital to analyze the effectiveness of the prescription given by its doctors. This feedback will help the hospital to get an idea of doctors' treatment reputation and reliability. The patient will give rating and review against the doctors address from which he got the prescription using give *GiveReview()* function. The review is the detailed feedback by the patient. Rating input is rated higher for the recommended doctors. The reviews and ratings are then stored in blockchain as shown in figure 4.2 and the motivation of the model is taken from [71].

The immutability and integrity of reviews must be maintained so that whenever the hospital searches for reviews and ratings of doctors, the reliable data can be retrieved. The hospital can view the existing reviews and ratings on doctors' prescriptions. If the review is stored in blockchain, the SC will return the review using *ReviewExist()* function otherwise it will return zero. The reviews and ratings can be retrieved using *SearchReview()* and *SearchRating()* functions, respectively.

4.2 Conclusion of the Chapter

In this chapter, we have explained the proposed techniques in detail. Additionally, smart contracts are explained for better understanding of the work done. In Chapter 6, the simulation results of the proposed solutions are discussedand the techniques are validatedand discussed. The thesis is then concluded in Chapter 7.

Chapter 5

Simulations and Results of Congestion Avoidance and Fault Detection using Data Science in WSN

5.1 Simulations and Results: Data Science in WSN

In section 5.1.1, proficiency of the proposed methods for congetion avoidance is evaluated in comparison with the other classification techniques. Section 5.1.4 provides the comparison of the proposed fault detection technique with other techniques. The simulation results section consists of the results based on both the original and prepared datasets. At first, the proficiency of the proposed GWO-SVM and DE-SVM is evaluated. At second, a comparison of GWO-SVM and DE-SVM with other classifiers is presented using three performance measures; Mean Square Error (MSE), Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). At third, the data learning rates are provided in a summarized form and a comparison of the proposed ERF with all other classification methods for faults detection based on two metrics is presented.

5.1.1 Original Dataset

The results based on original dataset for transmission rate adjustment are presented in this section. We have used the same data set provided in [4]. The original dataset consists of four features that are buffer occupancy ratio, congestion degree, transmission rate and amount of retransmission packets. Using the original dataset, proficiency of SVM is evaluated and the presented techniques are compared with other classification techniques like GA-SVM, Random Forest, SGD, MLP, NBand k-NN based on MSE, MAE and RMSE. In order to evaluate the performance of the proposed techniques,we have performed simulations in python 3.7. Specifications of the system used are: 1.61 GHz processor and 8.00 GB RAM.

5.1.2 Proficiency of proposed GWO-SVM and DE-SVM

We have taken total data of 400 inputs for simulations. The data used for training phase and test phase are 80% and 20%, respectively. The SVM parameters are tuned using the GWO and DE. Maximum iterations and number of search agents in GWO are taken as 50 and 5, respectively. Whereas in DE, mutation, crossover, population size and number of iterations are specified as 0.8, 0.7, 50 and 3, respectively.

A contingency table or confusion matrix is also calculated in order to get a glance of predictions. The error matrix gives the visualization of errors being made during classification by the classifier. All correct and incorrect predictions are specified in a matrix. The matrix consists of rows and columns which presents the instances in predicted and actual class. The proposed techniques are made using five SVMs, so the proposed confusion matrix is a matrix of five rows and five columns. The advantage of using the confusion matrix is to have a clear idea of what type of errors the classification model has made and how much data is predicted accurately. The confusion matrices C1 and C2 of the applied GWO-SVM and DE-SVM techniques are presented as follows:

$$C1 = \begin{bmatrix} 20 & 2 & 0 & 0 & 0 \\ 2 & 6 & 1 & 1 & 0 \\ 0 & 3 & 8 & 0 & 0 \\ 1 & 0 & 6 & 8 & 1 \\ 0 & 0 & 0 & 5 & 16 \end{bmatrix}$$

$$C2 = \begin{bmatrix} 20 & 2 & 0 & 0 & 0 \\ 3 & 6 & 1 & 0 & 0 \\ 0 & 3 & 8 & 0 & 0 \\ 1 & 0 & 6 & 8 & 1 \\ 0 & 0 & 0 & 6 & 15 \end{bmatrix}$$

The correctly predicted values are shown on the diagonals of the matrices. The values that are predicted more than the real dataand less than the real data are located as the upper and lower triangular elements of the matrices respectively. As shown in the error matrices, more than 70% data are located on the diagonal which means more than 70% data are accurately predicted. Upper triangular data shows higher transmission rate in the node. To conclude, the presented techniques correctly determined the amount of retransmission based on inputs as the predicted values are mostly correct.

5.1.3 Comparison of GWO-SVM and DE-SVM with other Classifiers

The overall error is calculated to evaluate the quality of the presented technique with other classification methods like GASVM, NB, RF, SGD, MLPand k-NN.

The data taken in training and testing phases are similar in all methods.

5.1.3.1 Genetic Algorithm based SVM

Classification is done using SVM and the parameters are tuned using GA. The adjusted parameters are penalty ratio, acceptable error in SVM and the deviation of the Gaussian kernel function. Implementation of GA is done using uniform crossover and mutation. The values used for population size, crossover and mutation are 50, 0.7 and 0.3 [4]. The confusion matrix of GA-SVM is presented as follows:

$$C3 = \begin{bmatrix} 22 & 0 & 0 & 0 & 0 \\ 3 & 4 & 3 & 0 & 0 \\ 0 & 1 & 9 & 1 & 0 \\ 1 & 0 & 6 & 7 & 2 \\ 0 & 0 & 1 & 5 & 15 \end{bmatrix}$$

From the confusion matric C3, we can deduce that GA-SVM showed more errors than our proposed methods and did not handle the complex data more accurately than our techniques.

5.1.3.2 Random Forest

RF [22] includes numerous different decision trees. Each decision tree analyses and votes on how the feature must be is classified. New items are classified based on voting done by the trees in the forests. We have applied random forest tree to solve the problem. Number of estimators and random state are taken as 9 and 42, respectively. The MSE, MAEand RMSE of RF are displayed in Figures 5.3, 5.4 and 5.5. The bar plots display that random forest works quite well on this data set. However, random forest did not produce as much accurate results as the proposed techniques. Figure 5.1 displays the tree generated using Weka tool. M5P is a well known binary regression model in which the the last nodes produce continous attributes. A standard deviation reduction or divergence metric is used to construct this tree.

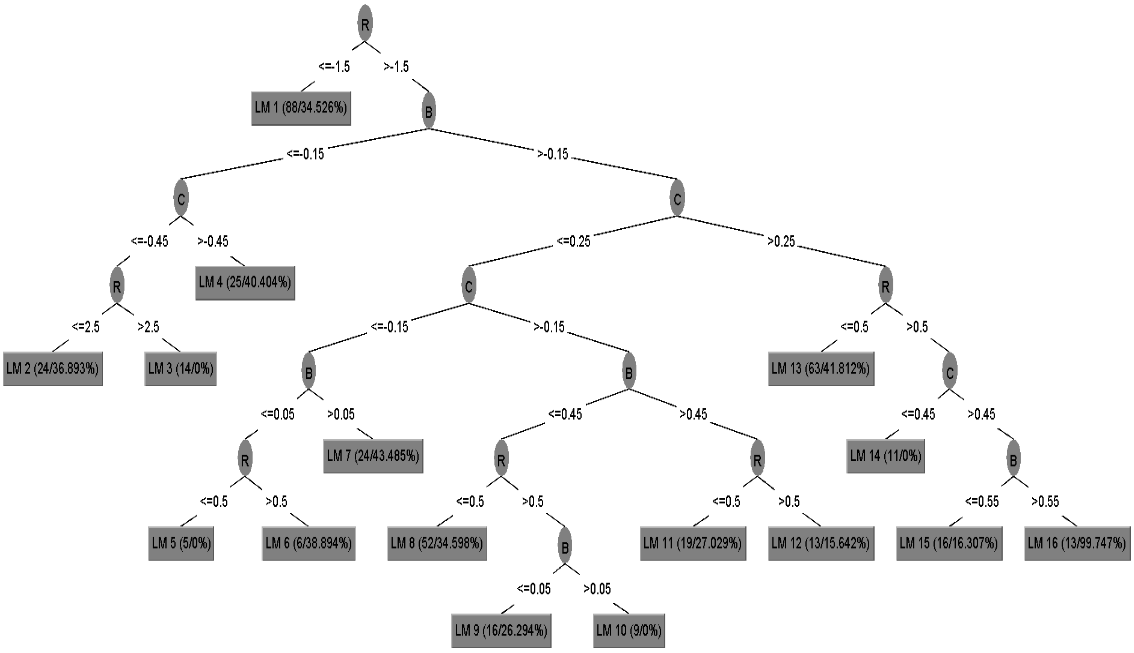

FIGURE 5.1: The Graph of M5P tree results

5.1.3.3 Naive Bayes

Naïve Bayes [22] is an efficient supervised learning algorithm that uses conditional probabilities to predict an outcome. It works accurate in real world scenarios. NB is based on statistics and assesses each feature independently in the data set. It deals with two features independently. In this way, a firm correlation between the factors is made. However, we have used Gaussian Naïve Bayes (GaussianNB) and checked the performance of this classifier on the given problem. This class (GaussianNB) assumes the features to be normally distributed. At first, we have scaled the features and then classified them using GaussianNB. The MSE, MAE and RMSE are displayed in figures 5.3,5.4 and 5.5, respectively. This classifier handles features independently and assumes that the presence of a feature is unassociated to the presence of other features. The results proved that naive bayes does not work well on the given dataset because provided features are related in our scenario i.e., transmission rate is dependent upon the the features of traffic loading information.

5.1.3.4 K-Nearest Neighbor

KNN is used to measure the difference based upon a distance function. It finds the closest neighbors of an instance and assigns a class to the instance based on voting [14]. The results depend upon the number of neighbors selected. We have randomly selected the number of neighbors and number of jobs as 5 and 2, respectively. The K value impacts the accuracy of the results. Figure 5.2 shows the error for the predicted values of test set for all the K values between 1 and 25. Figures 5.3,5.4 and 5.5 show that k-NN produced less classification errors as compared to RF, NB, SGD and MLP. However, k-NN did not perform well as compared to GA-SVM, GWO-SVM and DE-SVM since k-NN it can have high variance as it is not model based. The reason behind low testing performance of k-NN is that it overfits the data and produce unreliable training predictions of observations if the data is finite.

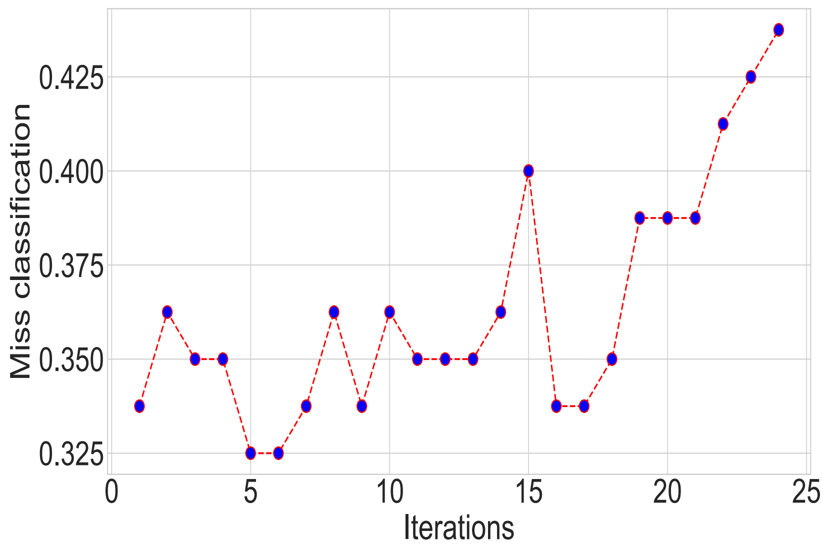

FIGURE 5.2: Miss classification in k-NN

TABLE 5.1: Errors of all Classifiers

Errors	GA-SVM	Random Forest	Naïve Bayes	k-NN	GWO-SVM	DE-SVM	SGD	MLP
MSE	0.425	0.463	0.664	0.458	0.412	0.387	4.4625	0.475
MAE	0.325	0.400	0.580	0.333	0.312	0.312	1.6875	0.375
RMSE	0.425	0.460	0.960	0.458	0.412	0.387	4.4625	0.475

5.1.3.5 Stochastic Gradient Descent

SGD [17] classifier is an optimization method that minimizes or maximizes a loss function. The gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule. SGD classifier requires less memory. It uses different parameters like Penalty, Loss (loss function i.e., hinge and squared_hinge) Dual (for optimization problem), Tol (for stopping criteria)and Random_state (to generate random number). The default setting of parameters in our scenario is taken as; loss=hinge,penalty=l2 with random state 7. Figures 5.3, 5.4 and 5.5 show the MSE, MAE and RMSE of SGD as 4.462, 1.687 and 4.462, respectively. The greater number of error means that SGD does not efficiently work on the given dataset and proved to be the worst classification method in our scenario. This is because the loss function taken here is lazy, it only updates the model parameters if an example violates the margin constraint. This makes training less efficient and may result in sparser models.

5.1.3.6 Multilayer Perceptron

MLP [18] falls in the category of feed forward ANN. MLP is a supervised learning technique. Like Artificial Neural Network (ANN), MLP has three layers as well. Each layer consists of neurons or nodes. Each node or neuron uses an activation function except for the input neurons. The activation functions here are nonlinear. This class of ANN uses backpropagation for training. We have set the regularization parameter as alpha=0.0001 and other parameters are specified as; hidden_layer_sizes=(100,100,100), learning_rate_init=0.1, verbose=10, solver=lbfgs, random_state=21 and max_iter=50. The MSE, MAE and RMSE of MLP are shown as 0.475, 0.475and 0.375 in figures 5.3, 5.4 and 5.5, respectively. MLP like other classifiers i.e., NB and SGD proved to be a bad classification method for the given observations.

Simulation results show that the proposed techniques GWO-SVM and DE-SVM outperform the other classifiers. DE-SVM has done classification more accurately with less errors. Proposed GWO-SVM presented the second best results and proved to be a good classification method in our scenario. As compared to other classification methods like GA-SVM, RF, NB and k-NN, DE-SVM is more robustand it performs computations efficiently. DE-SVM better classifies the continuous data and provides results faster whereas GA-SVM provides good enough results on discrete problems. Figures 5.3, 5.4 and 5.5 display the performance evaluation using MSE, MAE and RMSE of all the classifiers, respectively. The comparison results concluded that the proposed techniques GWO-SVM and DE-SVM solve the congestion problem and adjust the rate of transmission in a better way.

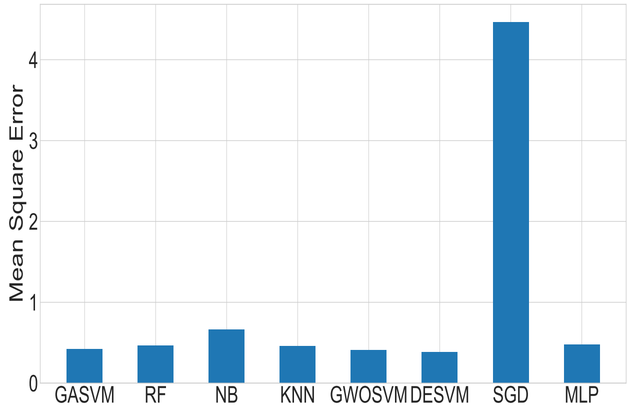

FIGURE 5.3: Classification Errors Comparison

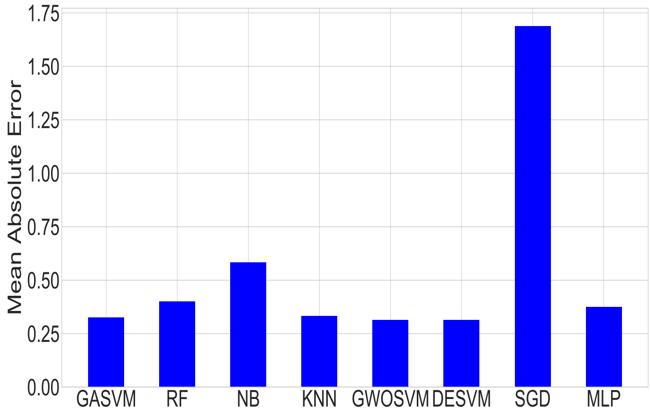

FIGURE 5.4: Classification Errors Comparison

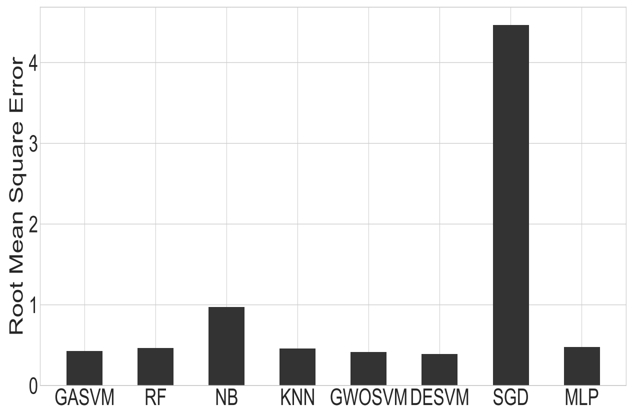

FIGURE 5.5: Classification Errors Comparison

5.1.4 Prepared Dataset

We have taken the dataset of sensor readings in a multi-hop WSNs from the published research [6]. We have prepared 20 new datasets by inducing four types of faults in the original data set. We have used the different classifiers for fault detection based on varying data learning rates. We have prepared new datasets that comprise of 3 features i.e., buffer occupation ratio, congestion degree and transmission rate. Then, we introduced a set of random faults. With different rates of faults; 50%, 40%, 30%, 20%and 10% and different types of data faults; offset, gain, stuck-at, out of bounds, we have prepared 20 datasets. These faults are induced based on the fault equations mentions above. Each dataset is divided into two parts i.e., training and testing. The learning phase uses 2/3 observations and the other 1/3 part is used for testing purpose. The faulty and non-faulty observations in the dataset are labeled as -1 and 1 respectively. We have presented ERF method to deal with faulty data. A comparative analysis is done to evaluate the efficiency of ERF as compared to GWO-SVM, DE-SVM, GA-SVM, k-NN, NB, MLP and SGD.

5.1.4.1 Evaluation Measures

The techniques can be evaluated using the criteria of; True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN), Detection Accuracy (DA) and using True Positive Rate (TPR) [46]. The detail is given as; TP refers to the correctly classified faulty cases. Whereas, FP presents the incorrectly classified faulty cases. Here, FPs are the inaccurate prediction of data being faulty. TN and FN refer to the normal cases and abnormal cases classified correctly and incorrectly, respectively. In our scenario, TNs are the correct indications of data being non-faulty. In the current work, we have considered only two metrics to perform comparative analysis of the proposed techniques. The first metric is Detection Accuracy (DA) and it is defined as [6]:

$$DA = \frac{\text{Faulty readings detected}}{\text{Total number of faulty readings}} \tag{5.1}$$

The second metric is True Positive Rate (TPR). It represents the correctly identified actual positives. The TPR is defined as [6]:

$$TPR = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.2}$$

In equation (8), correctly identified measurements are True Positive (TP) and False Negative (FN) are incorrectly rejected measurements [48]. A true positive is defined as an outcome where the model correctly predicted the readings [76]. It refers to the accurate predictions that were basically positive. In our scenario, TP is the true prediction of faults. A false negative is defined as a error including a test result improperly, indicates absence of a condition (the result is negative), when in reality it is present [77]. FN refers to the inaccurate predictions that are actually negative. Here, FN are the inaccurate predictions of data that is predicted as non-faulty.

5.1.4.2 Data Learning Rate Results

Table 5.2 shows the classification accuracies of different fault types in all classifiers.

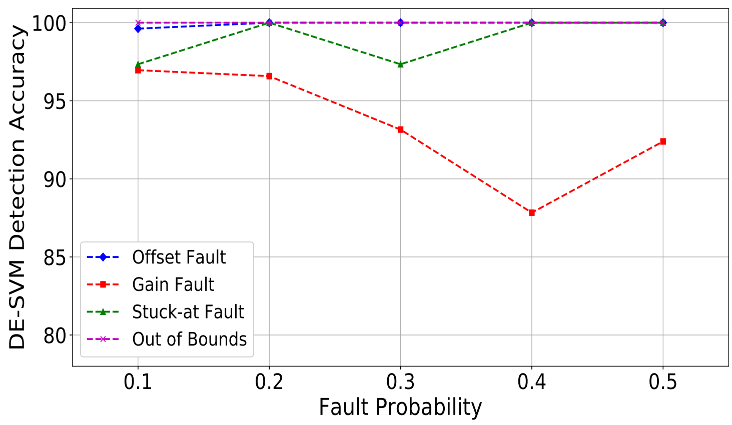

FIGURE 5.6: Detection Accuracy of DE-SVM on all Faults

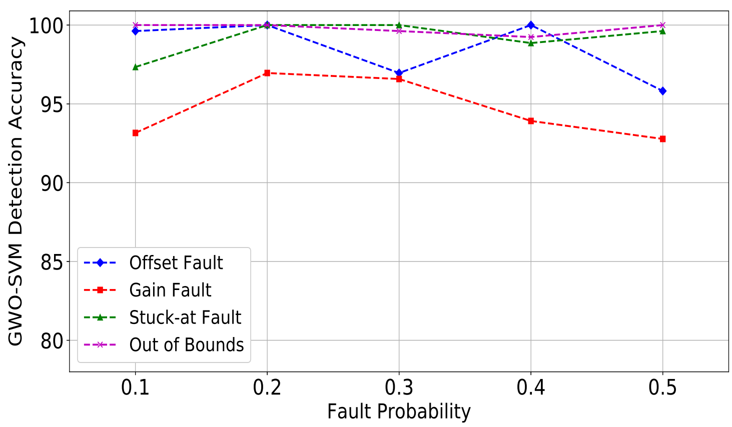

FIGURE 5.7: Detection Accuracy of GWO-SVM on all Faults

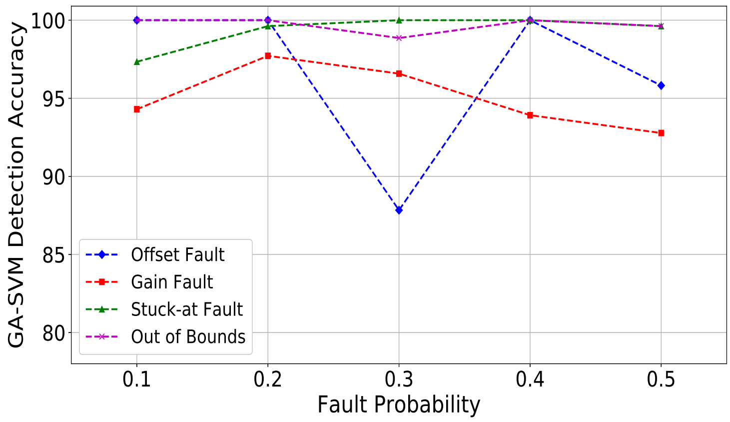

FIGURE 5.8: Detection Accuracy of GA-SVM on all Faults

Figures 5.6, 5.7 and 5.8 show the DA of the schemes DE-SVM, GWO-SVM and GA-SVM for all faults, i.e., offset, gain, stuck-at and out of bound. The SVM parameters are tuned using optimization algorithms to produce best learning rates. The rates at which four types of faults are induced are 10%, 20%, 30%, 40%, 50%. The fault rates are shown on x-axis whereas the detection accuracies of the techniques are shown on y-axis. It is clearly shown in Figure 5.6 that DE-SVM has detected the out of bound fault accurately with nearly 100 accuracy whereas there is a variation in detecting the gain fault at different rates. GWO-SVM has shown best accuracies in detecting stuck-at and out of bound faults whereas the gain faults detection is below 95 at 10, 40 and 50 percent fault rates. Briefly, the trends show that detection accuracies of gain fault at 0.4 rate of DE-SVM, GWO-SVM and GA-SVM are 87%, 93% and 93% respectively. However, the schemes show more accuracy in other three faults.

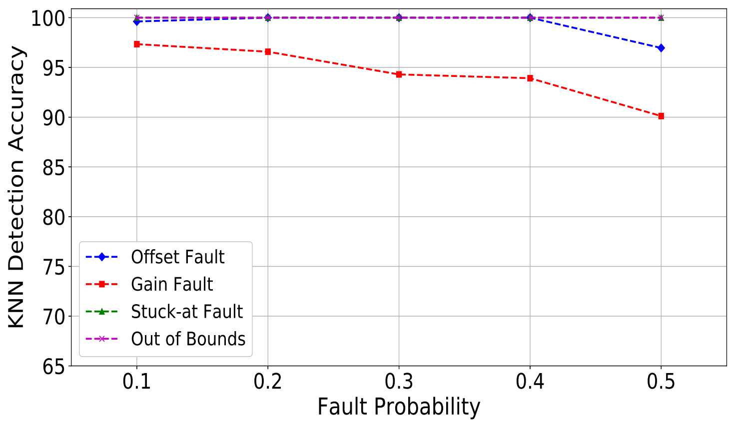

FIGURE 5.9: Detection Accuracy of k-NN on all Faults

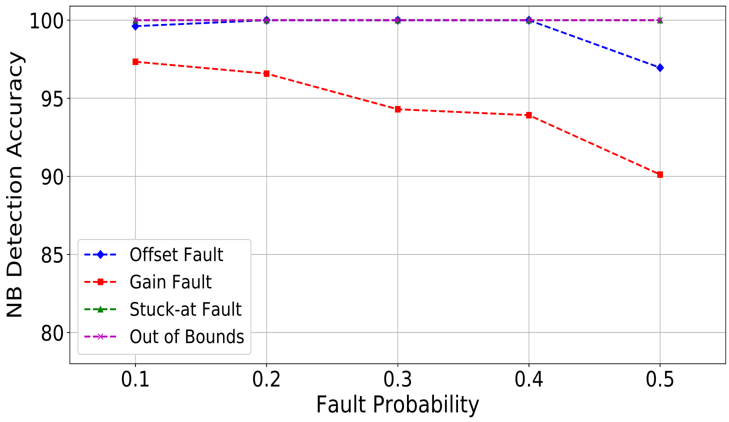

FIGURE 5.10: Detection Accuracy of NB on all Faults

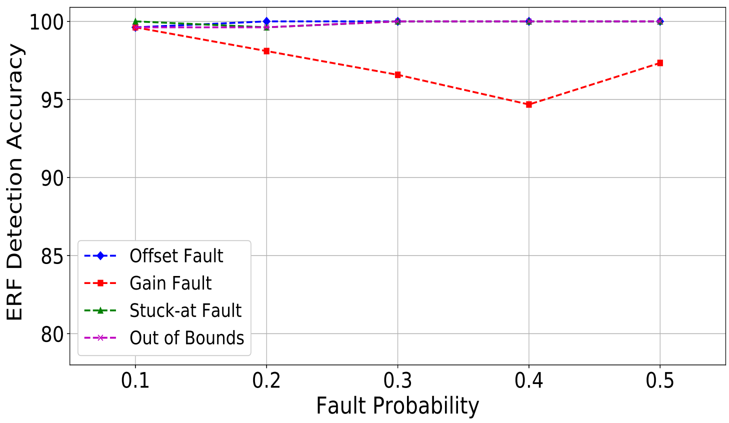

FIGURE 5.11: Detection Accuracy of ERF on all Faults

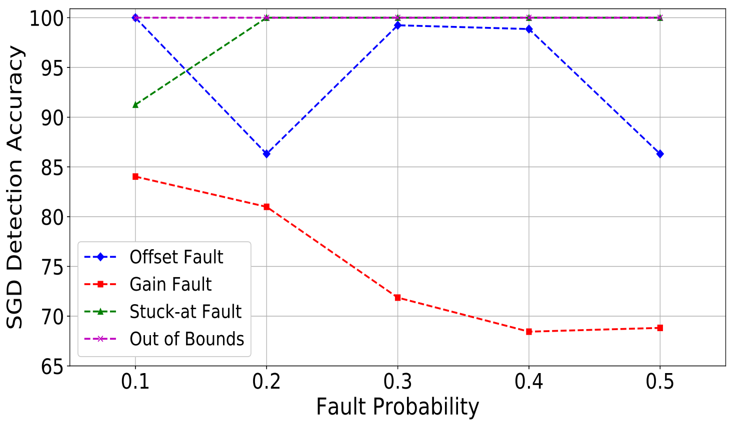

FIGURE 5.12: Detection Accuracy of SGD on all Faults

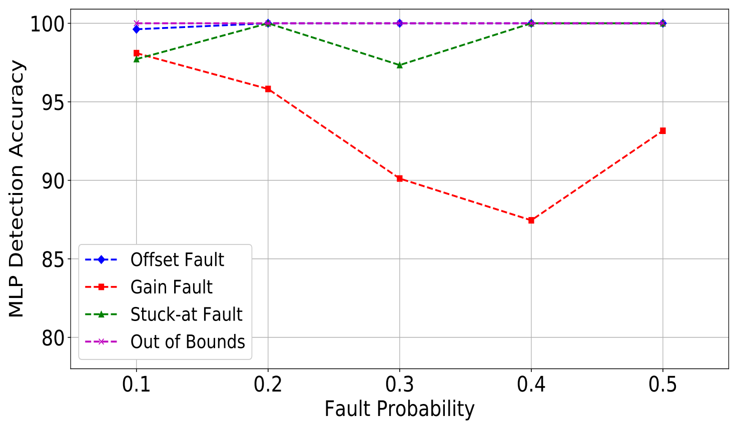

FIGURE 5.13: Detection Accuracy of MLP on all Faults

Figure 5.9 shows that, k-NN provides better results of offset, stuck-at and out of bound faults. However, it has produced slightly less accuracies for gain fault. The parameters for acquiring good results are set as leaf_size=30, metric=minkowski, n_jobs=2, n_neighbors=5, p=1and weights=uniform. The results validated the performance of k-NN in fault detection. The NB classifiers works quite well on the prepared (faulty) dataset as shown in Figure 5.10. We have scaled the data and then applied GaussianNB function to get accurate classification results. It can be noticed that k-NN and NB have shown nearly similar results and their detection rate is good enough. The trend in figure 5.11 shows the efficiency of ERF classifier for all type of faults. The parameters taken for are n_estimators = 5 and random_state = 42. The n-estimators represent the number of trees in a forest. When the number of trees is decreased, the classifier showed best possible results in our faulty dataset scenario. However, random state tuning did not impact much in the acquisition of best results. The result validates the working of ERF in fault detection. The ERF worked best in the detection of offset, stuck-at, out of bound faults expect for the gain fault. The gain fault accuracies of all techniques are not as good as the other faults. So, we can conclude that ERF has detected faults accurately and shown the best possible results than other techniques. Figure 5.12 presents the DA of SGD classifier of all fault types. The results vary in all faults at different fault rates. The reason behind this variation is that, we have not scaled the data before applying SGD. However, SGD gives better results on scaled and sparse data. The soft margin loss function hinge is used because if margin limit is violated, it updates the parameter, however, it is known to be a lazy function. The penalty is set as 12 and random state is taken as 7. The least accuracy in detection of gain fault is nearly

TABLE 5.2: Fault Detection Accuracies of Classifiers at different Fault Rates

Classifiers	**Fault Types**	**Rate 10%**	**Rate 20%**	**Rate 30%**	**Rate 40%**	**Rate 50%**
DE- SVM	Offset	99.61	100.0	100.0	100.0	100.0
	Gain	96.95	96.57	93.15	87.83	92.39
	Stuck-at	97.33	100.0	97.33	100.0	100.0
	Out of Bound	100.0	100.0	100.0	100.0	100.0
GWO- SVM	Offset	99.61	100.0	96.95	100.0	95.81
	Gain	93.15	96.95	96.57	93.91	92.0
	Stuck-at	97.33	100.0	98.85	99.61	99.00
	Out of Bound	100.0	100.0	99.61	99.23	100.0
GA- SVM	Offset	100.0	100.0	87.83	100.0	95.81
	Gain	94.29	97.71	96.57	93.91	92.77
	Stuck-at	97.33	99.61	100.0	100.0	99.61
	Out of Bound	100.0	100.0	98.85	100.0	99.61
***k*-NN**	Offset	95.43	94.29	90.87	91.25	93.53
	Gain	100.0	100.0	100.0	100.0	100.0
	Stuck-at	95.81	100.0	97.33	100.0	100.0
	Out of Bound	99.61	100.0	98.47	100.0	98.09
NB	Offset	97.33	96.57	94.29	93.91	90.11
	Gain	100.0	100.0	100.0	100.0	100.0
	Stuck-at	100.0	100.0	100.0	100.0	100.0
	Out of Bound	99.61	100.0	100.0	100.0	96.95
ERF	Offset	100.0	100.0	100.0	100.0	100.0
	Gain	99.61	98.85	96.95	94.67	96.57
	Stuck-at	100.0	100.0	100.0	100.0	100.0
	Out of Bound	100.0	100.0	100.0	100.0	100.0
SGD	Offset	100.0	86.31	99.23	98.85	86.31
	Gain	84.03	80.98	71.86	68.44	68.82
	Stuck-at	91.25	100.0	100.0	100.0	100.0
	Out of Bound	100.0	100.0	100.0	100.0	100.0
MLP	Offset	99.61	100.0	100.0	100.0	100.0
	Gain	98.09	95.81	90.11	87.45	93.15
	Stuck-at	97.71	100.0	97.33	100.0	100.0
	Out of Bound	100.0	100.0	100.0	100.0	100.0

65 and as the fault rates increases, the accuracy of SGD decreases. The variation in the detection at all rates shows the poor performance of SGD in fault detection. Figure 5.13 shows the detection accuracies of al faults using MLP. We have set the regularization parameter as alpha=0.0001 and other parameters are specified as hidden_layer_sizes=(100,100,100), learning_rate_init=0.1, verbose=10,

TABLE 5.3: Average Accuracies of all Classifiers

GWO-SVM	**DE-SVM**	**GA-SVM**	**SGD**	**MLP**	**ERF**	**NB**	***k*-NN**
0.70	0.65	0.71	0.22	0.66	0.81	0.35	0.66

solver=lbfgs, random_state=21, max_iter=50. MLP classifies the faults in a better way when we take alpha as 0.0001. MLP seems to be a good classification method than SGD in the given scenario. Clearly, the presented ERF outshines the other classifiers showing best quality results in terms of high accuracy. Average accuracies are presented in table 5.3.

Figure 5.14 shows the correctly identified values by all classifiers using the performance metric TPR. RF has performed better in identifying the faults induced in the original dataset readings. GWO-SVM, DE-SVM and GA-SVM identify the faults better than other techniques. SGD does not work well as compared to the other classification techniques and does not identify the faulty readings in a more efficient manner than other classifiers.

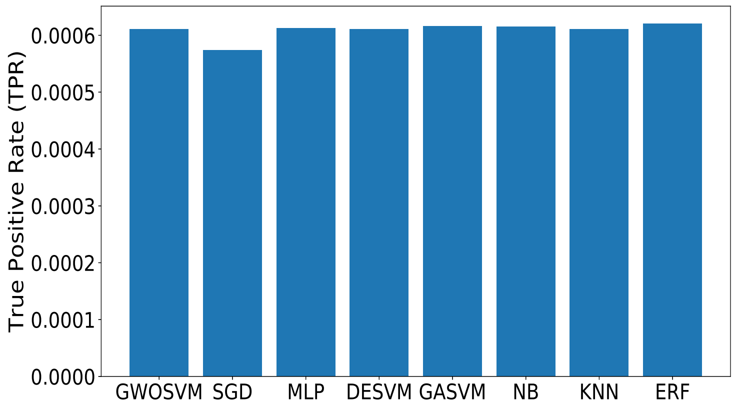

FIGURE 5.14: Average True Positive Rate

5.2 Conclusion of the Chapter

The detailed discussion of simulation results is given in this chapter. The adaptivity of the proposed works is then verified. Moreover, comparison with existing works is also done. In Chapter 7, the thesis is concluded alongwith future directions.

Chapter 6

Simulations and Results of Remote Patient Monitoring using Blokchain in IoT

6.1 Simulations and Results: Blockchain in IoT

In section 6.1.1, smart contracts validation is provided based on cost and time. The specifications of the system used are: CPU@1.61GHz, 8GB RAM, 64 bit operating system and X64-based processor. We have used solidity language for writing our SCs. We have used open source web browser environment Remix to test, debug and deploy our SCs.

6.1.1 Evaluation Metrics

The proposed system is evaluated using two metrics: cost and time. Results are discussed in detail below.

6.1.1.1 Cost

c Instructions are executed on every network node whenever a contract is implemented. The operations being executed have a cost that is stated in gas units. Whenever an ethereum transaction takes place on the blockchain, two types of costs are associated with it: one is the transaction cost and the other is execution cost. The blockchain network has the potential to increase trust by reducing the transaction costs because of its decentralized nature with no third party involved. Transaction cost includes the cost of data being sent, operations being performed and the storage of contract. Transaction cost is determined by gasUsed$\times$gasPrice where gasPrice is specified by the user and gasUsed refers to the total gas used for operations. Execution cost refers to the storage of local and global variables as well as the processing power for calculations.

Figure 6.1 shows the transaction and execution costs of all SCs. SCs are shown on the x-axis and their gas consumption on y-axis. Enrollments of patients and doctors shows the costs about 2692790 gas and 1986938 gas in transaction and execution of the contract. Monitoring and IoT device SCs cost less gas as compared to other contracts because the number of inputs fed to the monitoring contract are less than the inputs fields given in enrolments. More gas consumption in enrolments depicts a huge internal storage because the more data sent to the contract, the more cost it takes. Enterprise contract deployment took 1308577 as transaction and 950029 as execution cost. Less costs are recorded in the deployment of IoT device and monitoring contracts that shows that these contracts are logically less complex. IPFS stores only the hash of data and hash is independent of data size. This helped greatly in storing large amount of health records on IPFS. The deployment costs of IPFS SC are also shown in which only the hashes of data are stored in SC. The one time deployment costs of transaction and execution are recorded as 1963972 and 1456512 respectively. The costs of storage are reasonable for keeping data hash on IPFS. The review SC took 567908 transaction cost because it has only three functions and less amount of gas is used by transactions. The costs are shown in table 6.1 along with the ethers (USD) used in the deployment of all contracts.

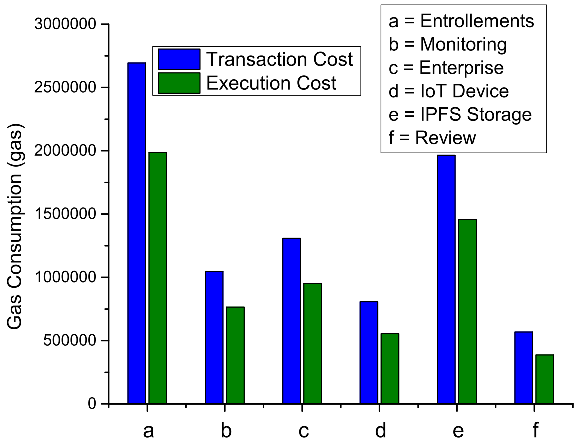

FIGURE 6.1: SCs Deployment

Figure 6.2 shows the subcontracts being called by the main monitoring contract on x-aix and the gas consumption on y-axis. The reason behind the deployment of six subcontracts is to check the amount of gas consumption for patients having more than 2 body sensors. These modular contracts cost less than the main contract because breaking the contract into subcontracts decreases the cost during interaction. There is a slight difference in all contracts costs because the modular concept makes the computation simpleand the data types used in all modular contract are almost same. However, the subcontract consuming the least transaction and execution gas is due to the reason that instances are using uint type instead of expensive types. This saves the blockchain from expensive storage of variables in terms of gas for a transaction. Modular contracts costs are shown in table 6.1.

Figure 6.3 displays the costs of transaction and execution taken by all functions of the enrolments SC. Adding the doctor and patient information cost about 236109 and 235845, respectively which is relatively high as compared to the costs of other functions. The execution costs of adding doctor and patient are recorded as 209333

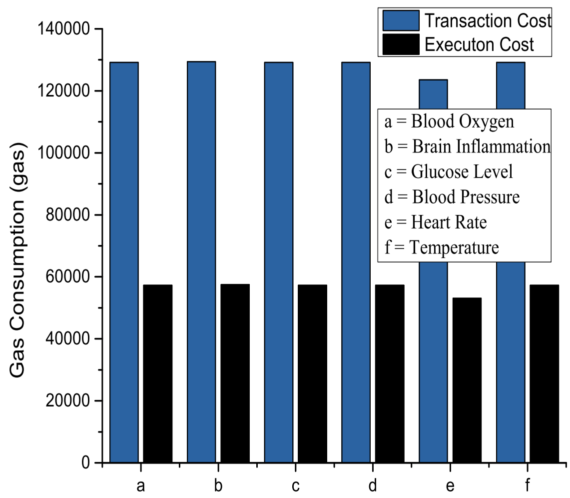

FIGURE 6.2: Patient Monitoring Modular SCs Deployment

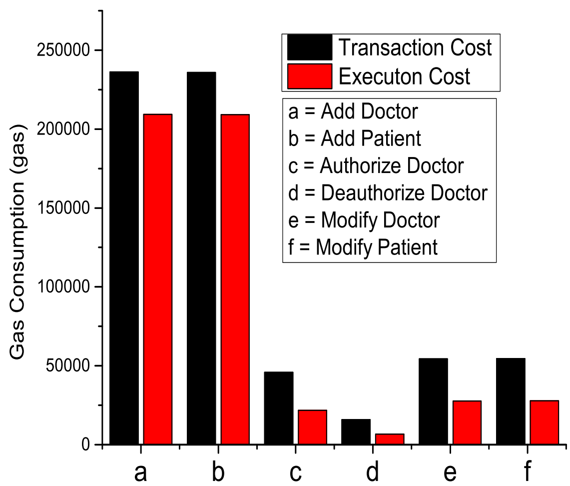

FIGURE 6.3: Enrollments Functions Costs

and 209069, respectively. The reason behind high costs is that the larger transactions require a huge amount of fee. Transaction costs of authorization, deauthorization, doctor modification and patient modification are 45832, 15788, 54365 and 54541, respectively. Execution costs of these four functions are 21744, 6700, 27589 and 27765, respectively. These functions consume less gas because smaller transactions are simpler to validate and consequently, consume less gas.

Figure 6.4 displays the gas consumption of IoT device contract where the device contract is createdand the possession is transferred from one custodian to the

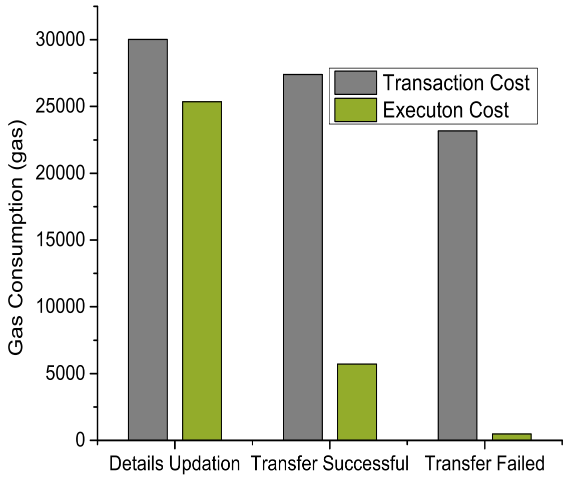

Figure 6.4: IoT Device Functions Costs

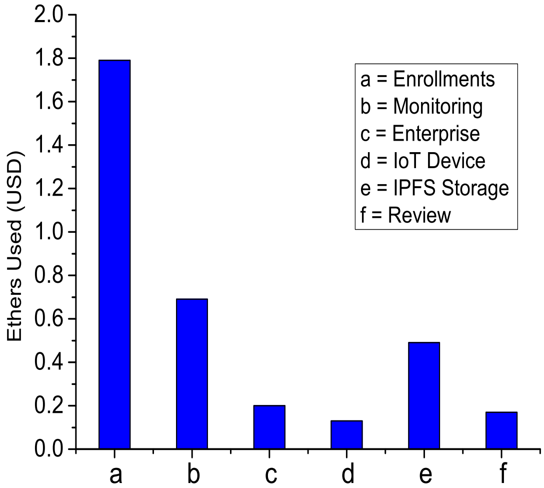

Figure 6.5: Ethers (USD) in Deployment of SCs

other. When the possession is transferred, new owner will be allowed to change the description of the device. The details are updated costing 30021 and 25357 as transaction and execution fee, respectively. The possession is successfully transferred consuming 27398 transaction cost whereas the failed transaction ended up consuming 23164 transaction cost. When the transfer is successful, the execution cost is recorded as 5710and if the same owner registers for the device again, the transfer is failed consuming 484 execution cost. Figure 6.5 shows the ethers used (converted to USD) on y-axis and all the SCs on x-axis. The figure shows total

USD we need for the deployment of our SCs in real-time environment. We need 0.69, 1.76, 0.20, 0.13 and 0.49 ethers (USD) for patient monitoring, enrolments, enterprise, IoT device and IPFS storage contracts, respectively when the current price of one ether is equal to 151.69 USD. All the transaction costs, execution costs, total ethers spent on transactions and ethers price in USD are given in table. 6.1. SCs execution cost corresponds to the processing time of transaction. There is a trade-off among transaction cost and transaction speed. For example, if we increase the transaction speed from slow to average and average to fast, the gas consumption will be increased. We will need to pay more cost if we need speedy transactions. Also, the fee consumption is effected by the length of input provided. The amount of ethers spent will be changed with every different input being fed to the contract.

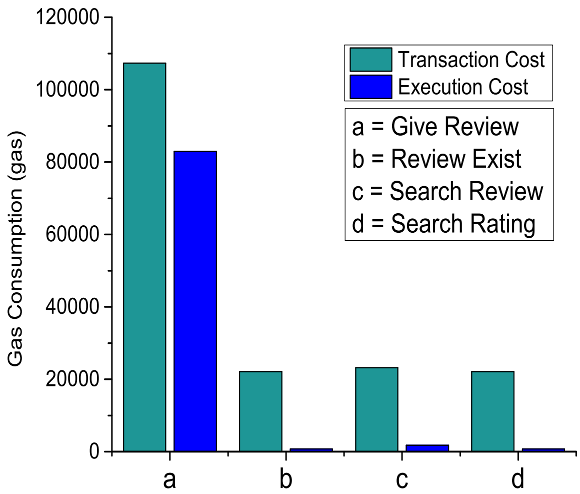

FIGURE 6.6: Rating and Review Functions

Figure 6.6 shows the transaction and execution costs of functions in review SC. The *GiveReview()* function takes 107330 transaction cost which is relatively high than *ReviewExist()*, *SearchReview()* and *SearchRating()* functions. This is due to the fact that the input given in GiveReview() is lengthy consisting of detailed review and rating. *ReviewExist()* function takes 22130 and 730 as transaction and execution cost, respectively. *ReviewExist()* only checks the review and rating and consequently consumes less gas. The transaction costs of *SearchReview()* and *SearchRating()* are 233180and 22112 whereas the execution costs are recorded to be 1780and 712. *SearchReview()*and *SearchRating()* functions consume the least gas because they only return the stored reviewsand ratings and computational overhead is less. Figure 6.7 shows the input length and the amount of gas consumed

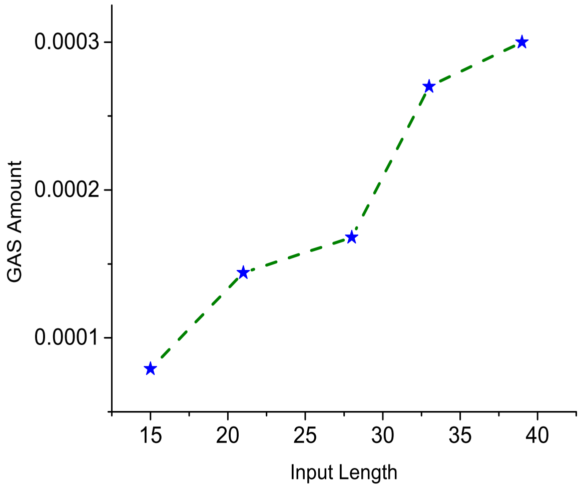

FIGURE 6.7: Input Length and GAS Amount

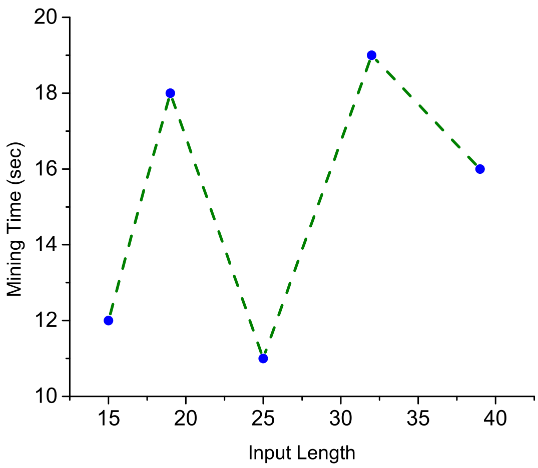

FIGURE 6.8: Input Length and Mining Time

on x-axis and y-axis, respectively. The gas consumption is directly proportional to the given amount of input. As shown, the gas consumption slightly increased with an increase in the input characters. Particularly, when the review is 39 characters long, the gas consumption is more as compared to the 33 characters long review. The graph in figure 6.8 shows the mining time on the y-axis against the string length displayed on x-axis. It can be observed that the mining time is not increased or decreased with the change in amount of characters being fed to the system. The mining time depends upon the network and is not affected by the input length.

TABLE 6.1: Deployment Costs

Contracts	**Transaction Cost**	**Execution Cost**	**Ether (USD)**
Monitoring	1047414	764994	0.69
Enrollments	2692790	1986938	1.76
Enterprise	1308577	950029	0.20
IoT Device	806517	554029	0.13
IPFS Storage	1963972	1456512	0.49
Review	567980	385824	0.17
Blood Oxygen	129164	57300	0.085
Brain Inflammation	129432	57500	0.085
Glucose Level	129164	57300	0.085
Blood Pressure	129164	57300	0.085
Heart Rate	123536	53100	0.081
Temperature	129164	57300	0.085

6.1.1.2 Time

- Encryption Time: The conversion from plaintext to ciphertext refers to encryption time of a technique. It depends on three factors; mode, plaintext and key size. We have made comparison of AES256 with other two encryption techniques in terms of execution time. The execution time is measured in milliseconds. The comparison is made to evaluate the efficiency of encryption algorithms. Figure 6.9 shows the recorded average execution time taken by affine cipher, AES256 and 3DES during encryption as 11.16, 6.25 and 9.08, respectively. In our experiment, the lowest time is consumed by AES256 that verifies its responsiveness. AES256 uses 2^{256} keys that makes it more secure. This feature serves the purpose as we need more secure algorithm due to the sensitivity of data. AES256 gives out the best performance and is more secure than other encryption algorithms. Other algorithms recorded more execution time and showed low performance.

- Decryption Time: Decryption time refers to the time taken to extract plaintext from ciphertext and it has a huge impact on performance of the algorithm. Figure 6.9 shows that the time taken by all algorithms for decryption is less than encryption time. The time consumed during decryption by affine cipher, AES256 and 3DES was 9.57, 2.74 and 4.81, respectively. AES256 has shown the best performance in decryption of the data hash because it is fast and secure, whereas 3DES gave satisfactory results. 3DES runs three times for encryption thereby taking more execution time. Affine cipher performed the worst in our scenario because it is slow and it needs more time to process the same amount of data as compared to other algorithms. After evaluating the encryption algorithms in terms of time, we implemented AES256 in IPFS smart contract as it takes the least amount of time. A comparison among traditional and blockchain systems is provided in table 6.2 to verify the effectiveness of using blockchain in healthcare.

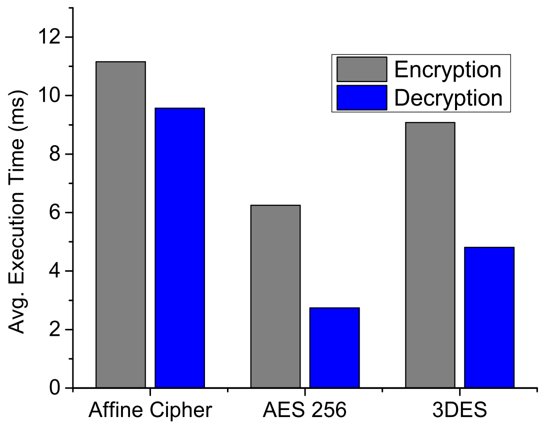

FIGURE 6.9: Average Execution Time of Encryption Techniques

TABLE 6.2: Comparison between Traditional and Blockchain Systems

Features	**Traditional Systems**	**Blockchain Systems**
Immutability	Unauthorized parties can hack and modify data	Creates unalterable logs of transactions
Availability	Data backups are maintained to deal with accidental cloud failures	Records are copied on all nodes and can be retrieved easliy
Transparency	Records are vulnerable to modifications	Changes in records can be traced easily
Integrity	Databases in the cloud server can be impaired by changes	All transactions can be restored using the merkle tree
Confidentiality and Privacy	Data transmission is made secure using encryption	Anonymity can be achieved using EOA where data associations with particular accounts are not easily determined thereby increasing security

6.2 Conclusion of the Chapter

The detailed discussion of simulation results is given in this chapter. The adaptivity of the proposed works is then verified. Moreover, comparison with existing works is also done. In Chapter 7, the thesis is concluded alongwith future directions.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

7.1.1 Congestion Avoidance and Fault Detection using Data Science in WSN

Many researches have alluded the efficiency of SVM classification method. This study aims to control congestion in WSNs by adjusting the transmission rate. Congestion degree and buffer occupancy ratio for different values of transmission rate are used to obtain the amount of retransmission packets. We have proposed two techniques namely DE-SVM and GWO-SVM to solve the problem of congestion and to classify the complex data. The simulation results show that the proposed DE-SVM and GWO-SVM efficiently deal with the complex data and outperforms the GA-SVM, k-NN, Naive Bayes, SGD, MLP and RF in terms of classification error. DE-SVM has given 3% and GWO-SVM has produced 1% less classification errors as compared to the state of the art techniques. We have injected four types of faults in the sensor data with different rates of faults and presented a comparative analysis of classifiers on the prepared datasets. We have proposed ERF for the detection of faults. Two metrics; Detection Accuracy and True Positive Rate are used for fault detection. The results show that the proposed ERF works well and classifies the faulty data in a more accurate manner. ERF has detected the faults with 81% accuracy that is more than the accuracies of other classifiers. Briefly, the three presented techniques outperformed the other classification methods. We plan to publish the prepared datasets of fault detection used in this thesis.

7.1.2 Remote Patient Monitoring using Blokchain in IoT

Remote medical care is rapidly increasing with an increase in the use of IoT devices. For improved health services, only the transfer of health status and patients' personal information is not enough rather an immutable record should be maintained. We have used blockchain for a secure and permanent log of health and personal data of patients. The unchangeable nature of blockchain enables us to keep track of unauthorized alterations to healthcare system. We have written SCs and provided patients and medical professionals with a secure way of enrolling themselves in a health centre. The health centre maintains the list of enrolled patients and authorizes them to doctors for treatment. The data gathered from patients' wearable medical sensors is used to remotely monitor the health conditions and is stored on IPFS after encryption. We have made comparison of three encryption techniques namely affine cipher, AES256 and 3DES. Out of these three, AES256 has taken the less execution time and proved to be more secure. Blockchain-based reviews are used to store the ratings given b patients on doctors' prescription. The medical device custody is verified through SCs and enabled the device custodian to transfer the possession of device to other patients. The results show the cost of all deployed contracts and the estimated amount of ethers in USD is provided to give readers an idea of the deployment cost in real time.

7.1.3 Future Work

7.1.4 Congestion Avoidance and Fault Detection using Data Science in WSN

For the future work, we aim to apply other optimization algorithms for parameter tuning of classification methods in order to get more accurate classification results in other scenarios. We will also introduce and classify more type of faults to decrease sensor failures and handle network traffic appropriately and we will consider more dynamic and practical test scenarios in WSNs.

7.1.5 Remote Patient Monitoring using Blokchain in IoT

For the future work, we aim to implement data sharing SCs for sharing patients personal and health data among various authorized health centres in order to reduce the chances of readmittance and also the treatment errors. For encryption, other techniques can also be used to make a more fair comparison.

Chapter 8

References

References

[1] Cui, S., Cao, Y., Sun, G. and Bin, S., **2018**. A new energy-aware wireless sensor network evolution model based on complex network. EURASIP Journal on Wireless Communications and Networking, p.218.

[2] Ren, Z.M., Zeng, A. and Zhang, Y.C., **2018**. Structure-oriented prediction in complex networks. Physics Reports, 750, pp.1-51.

[3] Shah, S.A., Nazir, B. and Khan, I.A., **2017**. Congestion control algorithms in wireless sensor networks: Trends and opportunities. Journal of King Saud University-Computer and Information Sciences, 29(3), pp.236-245.

[4] Gholipour, M., Haghighat, A.T. and Meybodi, M.R., **2017**. Hop by Hop Congestion Avoidance in wireless sensor networks based on genetic support vector machine. Neurocomputing, 223, pp.63-76.

[5] Muhammed, T. and Shaikh, R.A., **2017**. An analysis of fault detection strategies in wireless sensor networks. Journal of Network and Computer Applications, 78, pp.267-287.

[6] Zidi, S., Moulahi, T. and Alaya, B., **2017**. Fault detection in wireless sensor networks through SVM classifier. IEEE Sensors Journal, 18(1), pp.340-347.

[7] Dwivedi, A.D., Srivastava, G., Dhar, S. and Singh, R., **2019**. A decentralized privacy-preserving healthcare blockchain for iot. Sensors, 19(2), p.326.

[8] Griggs, K.N., Ossipova, O., Kohlios, C.P., Baccarini, A.N., Howson, E.A. and Hayajneh, T., **2018**. Healthcare blockchain system using smart contracts for secure automated remote patient monitoring. Journal of medical systems, 42(7), p.130.

[9] Al-Aboody, N.A. and Al-Raweshidy, H.S., **2016**, September. Grey wolf optimization-based energy-efficient routing protocol for heterogeneous wireless sensor networks. In Computational and Business Intelligence (ISCBI), 2016 4th International Symposium on (pp. 101-107). IEEE.

[10] Hafiza Syeda Zainab Kazmi, Nadeem Javaid, Muhammad Imran and Fatma Outay, **2019**. Congestion Control in Wireless Sensor Networks based on Support Vector Machine, Grey Wolf Optimization and Differential Evolution. 11th Wireless Days Conference (WD).

[11] Kazmi, H.S.Z., Nazeer, F., Mubarak, S., Hameed, S., Basharat, A., Javaid, N., **2019**. Trusted Remote Patient Monitoring using Blockchain-based Smart Contracts. In 14-th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA).

[12] Rajakumar, R., Amudhavel, J., Dhavachelvan, P. and Vengattaraman, T., **2017**. GWO-LPWSN: Grey wolf optimization algorithm for node localization problem in wireless sensor networks. Journal of Computer Networks and Communications, 2017.

[13] Tran, K.P. and Huong, T.T., **2017**, October. Data driven hyperparameter optimization of one-class support vector machines for anomaly detection in wireless sensor networks. In Advanced Technologies for Communications (ATC), 2017 International Conference on (pp. 6-10). IEEE.

[14] Deng, F., Guo, S., Zhou, R. and Chen, J., **2017**. Sensor multifault diagnosis with improved support vector machines. IEEE Transactions on Automation Science and Engineering, 14(2), pp.1053-1063.

[15] Kaur, R. and Arora, S., **2017**. Nature inspired range based wireless sensor node localization algorithms. International Journal of Interactive Multimedia and Artificial Intelligence, 4(6), pp.7-17.

[16] Shieh, C.S., Sai, V.O., Lee, T.F., Le, Q.D. and Lin, Y.C., **2017**. Node Localization in WSN using Heuristic Optimization Approaches.

[17] Miao, X., Liu, Y., Zhao, H. and Li, C., **2018**. Distributed Online One-Class Support Vector Machine for Anomaly Detection Over Networks. IEEE Transactions on Cybernetics, (99), pp.1-14.

[18] Swain, R.R., Khilar, P.M. and Bhoi, S.K., **2018**. Heterogeneous fault diagnosis for wireless sensor networks. Ad Hoc Networks, 69, pp.15-37.

[19] Gu, X., Deng, F., Gao, X. and Zhou, R., **2018**. An Improved Sensor Fault Diagnosis Scheme Based on TA-LSSVM and ECOC-SVM. Journal of Systems Science and Complexity, 31(2), pp.372-384.

[20] Yun, S., Lee, J., Chung, W., Kim, E. and Kim, S., **2009**. A soft computing approach to localization in wireless sensor networks. Expert Systems with Applications, 36(4), pp.7552-7561.

[21] Sujatha, S. and Siddappa, M., **2017**. Node localization method for wireless sensor networks based on hybrid optimization of particle swarm optimization and differential evolution. IOSR J Comput Eng, 19(2), pp.07-12.

[22] Goeschel, K., **2016**, March. Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive Bayes for off-line analysis. In SoutheastCon, 2016 (pp. 1-6). IEEE

[23] Gupta, S., Mittal, M. and Padha, A., **2017**, December. Predictive Analytics of Sensor Data Based on Supervised Machine Learning Algorithms. In 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS) (pp. 171-176). IEEE.

[24] Wang, Y., Yang, A., Chen, X., Wang, P., Wang, Y. and Yang, H., **2017**. A deep learning approach for blind drift calibration of sensor networks. IEEE Sensors Journal, 17(13), pp.4158-4171.

[25] Zhang, Y., Liu, Y., Chao, H.C., Zhang, Z. and Zhang, Z., **2018**. Classification of Incomplete Data Based on Evidence Theory and an Extreme Learning Machine in Wireless Sensor Networks. Sensors, 18(4), p.1046.

[26] Abuassba, A.O., Zhang, D., Luo, X., Shaheryar, A. and Ali, H., **2017**. Improving Classification Performance through an Advanced Ensemble Based Heterogeneous Extreme Learning Machines. Computational intelligence and neuroscience, 2017.

[27] Jafarizadeh, V., Keshavarzi, A. and Derikvand, T., **2017**. Efficient cluster head selection using Naïve Bayes classifier for wireless sensor networks. Wireless Networks, 23(3), pp.779-785.

[28] Jiang, C., Zhang, H., Ren, Y., Han, Z., Chen, K.C. and Hanzo, L., **2017**. Machine learning paradigms for next-generation wireless networks. IEEE Wireless Communications, 24(2), pp.98-105.

[29] Atiga, J., Mbarki, N.E., Ejbali, R. and Zaied, M., **2018**, April. Faulty node detection in wireless sensor networks using a recurrent neural network. In Tenth International Conference on Machine Vision (ICMV 2017) (Vol. 10696, p. 106962P). International Society for Optics and Photonics.

[30] Swain, R.R. and Khilar, P.M., **2017**. Composite fault diagnosis in wireless sensor networks using neural networks. Wireless Personal Communications, 95(3), pp.2507-2548.

[31] Yuvaraja, M. and Sabrigiriraj, M., **2017**. Fault detection and recovery scheme for routing and lifetime enhancement in WSN. Wireless Networks, 23(1), pp.267-277.

[32] Gao, Y., Xiao, F., Liu, J. and Wang, R., **2018**. Distributed Soft Fault Detection for Interval Type-2 Fuzzy-model-based Stochastic Systems With Wireless Sensor Networks. IEEE Transactions on Industrial Informatics.

[33] Zhang, D., Qian, L., Mao, B., Huang, C., Huang, B. and Si, Y., **2018**. A Data-Driven Design for Fault Detection of Wind Turbines Using Random Forests and XGboost. IEEE Access, 6, pp.21020-21031.

[34] Jan, S.U., Lee, Y.D., Shin, J. and Koo, I., **2017**. Sensor fault classification based on support vector machine and statistical time-domain features. IEEE Access, 5, pp.8682-8690.

[35] Shakeel, U., Jan, N., Qasim, U., Khan, Z.A. and Javaid, N., **2016**, July. DRADS: Depth and reliability aware delay sensitive routing protocol for underwater WSNs. In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2016 10th International Conference on (pp. 78-83). IEEE.

[36] Javaid, N., Majid, A., Sher, A., Khan, W. and Aalsalem, M., **2018**. Avoiding Void Holes and Collisions with Reliable and Interference-Aware Routing in Underwater WSNs. Sensors, 18(9), p.3038.

[37] Ali, B., Sher, A., Javaid, N., Aurangzeb, K. and Haider, S.I., **2018**. Retransmission avoidance for reliable data delivery in underwater WSNs. Sensors, 18(1), p.149.

[38] Ahmed, F., Wadud, Z., Javaid, N., Alrajeh, N., Alabed, M.S. and Qasim, U., **2018**. Mobile Sinks Assisted Geographic and Opportunistic Routing Based Interference Avoidance for Underwater Wireless Sensor Network. Sensors, 18(4), p.1062.

[39] Javaid, N., Shakeel, U., Ahmad, A., Alrajeh, N., Khan, Z.A. and Guizani, N., **2019**. DRADS: depth and reliability aware delay sensitive cooperative routing for underwater wireless sensor networks. Wireless Networks, 25(2), pp.777-789.

[40] Hadj-Mabrouk, H., **2019**. Contribution of artificial intelligence and machine learning to the assessment of the safety of critical software used in railway transport. AIMS Electronics and Electrical Engineering, 3(1), pp.33-70.

[41] Otoum, S., Kantarci, B. and Mouftah, H.T., **2019**. On the feasibility of deep learning in sensor network intrusion detection. IEEE Networking Letters, 1(2), pp.68-71.

[42] Otoum, S., Kantarci, B. and Mouftah, H.T., **2017**, June. Mitigating False Negative intruder decisions in WSN-based Smart Grid monitoring. In 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC) (pp. 153-158). IEEE.

[43] Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. and Caicedo, O.M., **2018**. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. Journal of Internet Services and Applications, 9(1), p.16.

[44] Sunitha, G.P., Kumar, B.V. and Kumar, S.D., **2018.** A Nature Inspired Optimal Path Finding Algorithm to Mitigate Congestion in WSNs. International Journal of Scientific Research in Network Security and Communication, 6(3), pp.50-57.

[45] Parsavand, H. and Ghaffari, A., **2018**. Controlling Congestion in Wireless Sensor Networks Through Imperialist Competitive Algorithm. Wireless Personal Communications, 101(2), pp.1123-1142.

[46] Aloqaily, Moayad, Balasubramanian, V., Zaman, F., Al Ridhawi, I., Jararweh, Y. **2018**. Congestion mitigation in densely crowded environments for augmenting qos in vehicular clouds." In Proceedings of the 8th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications, pp. 49-56. ACM.

[47] Al-khafajiy, Mohammed, Baker, T., Al-Libawy, H., Maamar, Z., Aloqaily, M., Jararweh, Y. **2019**. Improving fog computing performance via fog-2-fog collaboration. Future Generation Computer Systems 100: 266-280.

[48] Chen, W., Zhao, H., Li, T. and Liu, Y., **2018**. Optimal probabilistic encryption for distributed detection in wireless sensor networks based on immune differential evolution algorithm. Wireless Networks, 24(7), pp.2497-2507.

[49] Jia, B., Zhou, T., Li, W., Liu, Z. and Zhang, J., **2018**. A Blockchain-Based Location Privacy Protection Incentive Mechanism in Crowd Sensing Networks. Sensors, 18(11), p.3894.

[50] Zhang, Y. and Wen, J., **2017**. The IoT electric business model: Using blockchain technology for the internet of things. Peer-to-Peer Networking and Applications, 10(4), pp.983-994.

[51] Xu, C., Wang, K., Li, P., Guo, S., Luo, J., Ye, B. and Guo, M., **2018**. Making big data open in edges: A resource-efficient blockchain-based approach. IEEE Transactions on Parallel and Distributed Systems.

[52] Lin, D. and Tang, Y., **2018**. Blockchain Consensus Based User Access Strategies in D2D Networks for Data-Intensive Applications. IEEE Access, 6, pp.72683-72690.

[53] Jiang, T., Fang, H. and Wang, H., **2018**. Blockchain-based Internet of vehicles: distributed network architecture and performance analysis. IEEE Internet of Things Journal.

[54] Sharma, P.K., Moon, S.Y. and Park, J.H., **2017**. Block-VN: A Distributed Blockchain Based Vehicular Network Architecture in Smart City. JIPS, 13(1), pp.184-195.

[55] Zhang, G., Li, T., Li, Y., Hui, P. and Jin, D., **2018**. Blockchain-Based Data Sharing System for AI-Powered Network Operations. Journal of Communications and Information Networks, 3(3), pp.1-8.

[56] Novo, O., **2018**. Scalable Access Management in IoT using Blockchain: a Performance Evaluation. IEEE Internet of Things Journal.

[57] Sharma, P.K., Singh, S., Jeong, Y.S. and Park, J.H., **2017**. Distblocknet: A distributed blockchains-based secure sdn architecture for iot networks. IEEE Communications Magazine, 55(9), pp.78-85.

[58] Suliman, A., Husain, Z., Abououf, M., Alblooshi, M. and Salah, K., **2018**. Monetization of IoT data using smart contracts. IET Networks, 8(1), pp.32-37.

[59] Lin, J., Shen, Z., Miao, C. and Liu, S., **2017**. Using blockchain to build trusted lorawan sharing server. International Journal of Crowd Science, 1(3), pp.270-280.

[60] Xu, Y., Wang, G., Yang, J., Ren, J., Zhang, Y. and Zhang, C., **2018**. Towards Secure Network Computing Services for Lightweight Clients Using Blockchain. Wireless Communications and Mobile Computing, 2018.

[61] Qu, C., Tao, M., Zhang, J., Hong, X. and Yuan, R., **2018**. Blockchain based credibility verification method for IoT entities. Security and Communication Networks, 2018.

[62] Hammi, M.T., Hammi, B., Bellot, P. and Serhrouchni, A., **2018**. Bubbles of Trust: A decentralized blockchain-based authentication system for IoT. Computers and Security, 78, pp.126-142.

[63] Samuel, O., Javaid, N., Awais, M., Ahmed, Z., Imran, M., Guizani, M., **2019**. A Blockchain Model for Fair Data Sharing in Deregulated Smart Grids. In IEEE Global Communications Conference (GLOBCOM).

[64] Park, J.S., Youn, T.Y., Kim, H.B., Rhee, K.H. and Shin, S.U., **2018**. Smart contract-based review system for an IoT data marketplace. Sensors, 18(10), p.3577.

[65] Ding, S., Cao, J., Li, C., Fan, K. and Li, H., **2019**. A Novel Attribute-Based Access Control Scheme Using Blockchain for IoT. IEEE Access, 7, pp.38431-38441.

[66] Rehman, M., Javaid, N., Awais, M., Imran, M., Naseer, N., **2019**. Cloud based Secure Service Providing for IoTs using Blockchain. In IEEE Global Communications Conference (GLOBCOM).

[67] Zhang, A. and Lin, X., **2018**. Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain. Journal of medical systems, 42(8), p.140.

[68] Han, S.H., Kim, J.H., Song, W.S. and Gim, G.Y., **2019**. An empirical analysis on medical information sharing model based on blockchain. International Journal of Advanced Computer Research, 9(40), pp.20-27.

[69] Nizamuddin, N., Salah, K., Azad, M.A., Arshad, J. and Rehman, M.H., **2019**. Decentralized document version control using ethereum blockchain and IPFS. Computers and Electrical Engineering, 76, pp.183-197.

[70] Pãnescu, A.T. and Manta, V., **2018**. Smart Contracts for Research Data Rights Management over the Ethereum Blockchain Network. Science and Technology Libraries, 37(3), pp.235-245.

[71] Park, J.S., Youn, T.Y., Kim, H.B., Rhee, K.H. and Shin, S.U., **2018**. Smart contract-based review system for an IoT data marketplace. Sensors, 18(10), p.3577.

[72] Wu, S. and Du, J., **2019**, January. Electronic medical record security sharing model based on blockchain. In Proceedings of the 3rd International Conference on Cryptography, Security and Privacy (pp. 13-17). ACM.

[73] Košťál, K., Helebrandt, P., Belluš, M., Ries, M. and Kotuliak, I., **2019**. Management and Monitoring of IoT Devices Using Blockchain. Sensors, 19(4), p.856.

[74] Tian, H., He, J. and Ding, Y., **2019**. Medical Data Management on Blockchain with Privacy. Journal of medical systems, 43(2), p.26.

[75] Rahmadika, S. and Rhee, K.H., **2018**. Blockchain technology for providing an architecture model of decentralized personal health information. International Journal of Engineering Business Management, 10, p.1847979018790589.

[76] Aloqaily, M., Otoum, S., Al Ridhawi, I. and Jararweh, Y., **2019**. An intrusion detection system for connected vehicles in smart cities. Ad Hoc Networks, 90, p.101842.

[77] https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative [Last visit 1 Augest, 2019].

Appendices

.1 Detail of appendices

This section presents the implementation details of the thesis entitled "Towards Energy Efficiency and Trustfulness in Complex Networks Using Data Science Techniques and Blockchain".

This code is developed by Hafiza Syeda Zainab Kazmi under the supervision of Dr. Nadeem Javaid. To execute the code for Congestion Avoidance and Fault Detection in Chapter 3, copy the code from Appendix B and paste in PYTHON file and save it with .py extension. To execute the code for Blockchain in Chapter 4, copy the code from Appendix C and paste in REMIX IDE using the names given at the start of each function with .sol extension. If you need any help or have any query regarding the code execution, you can email me at zainab.kazmi13@gmail.com.
Email address: nadeemjavaidqau@gmail.com, zainab.kazmi13@gmail.com
The detail of appendices are as follows.

1 appendix B contains the PYTHON code for Congestion Avoidance and Fault Detection and

2 appendix C contains the REMIX code for Blockchain.

Readme:

- For problem 1:

1. please read and understand working of the algorithms in details,
2. copy the code and paste it in PYTHON file,
3. save the PYTHON file with an appropriate file name and .py extension as mentioned in code.

- For problem 2:

1. please read and understand working of the REMIX IDE in details,
2. copy the code and paste it in REMIX (online) file,
3. save the REMIX file with an appropriate file name and .sol extension as mentioned in code,
4. enter the inputs and run the contract.

.2 Implementation of Proposed Solution 1:

This code is developed by Hafiza Syeda Zainab Kazmi under the supervision of Dr. Nadeem Javaid. To execute the code for Congestion Avoidance and Fault Detection in Chapter 3, copy the code from Appendix B and paste in PYTHON file and save it with .py extension. If you need any help or have any query regarding the code execution, you can email me at zainab.kazmi13@gmail.com. You can find detailed guidelines in *readme.txt* file.

```

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sklearn
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn import linear_model
from sklearn.svm import SVR
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import KFold
from sklearn.cross_validation import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.model_selection import train_test_split #Import function for
    sliptting data into training and testing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt; plt.rcdefaults()
from random import random
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from pyeasyga import pyeasyga
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
sklearn.neighbors.DistanceMetric

```

Transmission Rate Adjustment

```
features = pd.read_csv( 'Dataset1.csv', skiprows=0,usecols=range(0,4))
features.describe()
print(features)
Y=features['RT']
X= features.drop('RT', axis = 1)
Y = np.array(Y)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.20,
    random_state=7)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

GWO-SVM

```
%%%%%%%%%%  ----- GWO-SVM-----%%%%%%%%%%%%%%%
Cs = [1, 1.99, 1.1,1.89, 1]
gammas = [1.9,1.7, 1.5, 3,1]
tol=[0.001,.01,.1,.01,2] #MSE=.4125
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
a=clf.score(x_test,y_test)
print ('Accuracy of GWOSVM',a)
print('Confusion matrix GWOSVM')
cm = confusion_matrix(y_test,y_pred)
print (cm)
print('GWo-SVM graphs')
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.plot(y_test,'-b',label='Test Data')
plt.ylim(0,4)
plt.plot(y_pred,'-r',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TestData','TestNet'), frameon=True)
plt.show();
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.ylim(0,4)
plt.plot(y_train[0:80] ,'-g',label='TrainData')
plt.plot(y_pred,'-b',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TrainData','TrainNet'), frameon=True)
plt.show();
MSEGWO=metrics.mean_squared_error(y_test,y_pred)
```

```
def MAPE(y_test, y_pred):                    # Formula of MAPE used
y_test=abs(y_test);y_pred=abs(y_pred);
y_test, y_pred = np.array(y_test), np.array(y_pred)
 return np.mean(np.abs((y_test - y_pred) / y_test)) * 100
MapeErr1=MAPE(y_test, y_pred)
MAEerr1=mean_absolute_error(y_test, y_pred);
print('MAPE of GWOSVM',MapeErr1)
print('MAE of GWOSVM',MAEerr1)
print('Accuracy of GWOSVM',100-MapeErr1)
def rmse(y_pred, y_test):
    differences = y_pred - y_test
    differences_squared = differences ** 2
    mean_of_differences_squared = differences_squared.mean()
    rmse_val = np.sqrt(mean_of_differences_squared)
    return rmse_val
MSEerror_GWOSVM= mean_squared_error(y_test, y_pred)
print('RMSEerror of GWOSVM',MSEerror_GWOSVM)
```

DE-SVM

```
Cs = [1,10,100,1000,10000]
gammas = [.1,.01,.001,.0001,.00001]
tol=[0.001,.01,.1,.01,2]
param_grid = {'C': Cs, 'gamma' : gammas}
clf = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
print(y_pred)
cm = confusion_matrix(y_test,y_pred)
print ('Confusion matrix  of DE-SVM',cm)
b=accuracy_score(y_test, y_pred)
print(a)
print ('accuDESVM',accuracy_score(y_test, y_pred))
print('DESVM Plots')
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.plot(y_test,'-b',label='Test Data')
plt.ylim(0,4)
plt.plot(y_pred,'-r',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TestData','TestNet'), frameon=True)
plt.show();
print('DESVM Plot')
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.xlim(0,80)
plt.ylim(0,4)
plt.plot(y_train,'-g',label='TrainData')
```

```
plt.plot(y_pred,'-b',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TrainData','TrainNet'), frameon=True)
plt.show();
MSE6=metrics.mean_squared_error(y_test,y_pred)
print('MSE of DE-SVM',MSE6)
width=1/4
plt.ylabel('Mean Square Error')
plt.ylim(0,0.7)
plt.bar(['DESVM'],[MSE6], width, align='center', alpha=0.7)
plt.show()
def MAPE(y_test, y_pred):
    y_test=abs(y_test);y_pred=abs(y_pred);
    y_test, y_pred = np.array(y_test), np.array(y_pred)
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100
from sklearn.metrics import mean_absolute_error
MapeErr6=MAPE(y_test, y_pred)
MAEerr6=mean_absolute_error(y_test, y_pred);
print('MAPE of DE-OSVM',MapeErr6)
print('MAE of DE-SVM',MAEerr6)
print('Accuracy of DE-SVM',100-MapeErr6)
def rmse(y_pred, y_test):
    differences = y_pred - y_test
    differences_squared = differences ** 2
    mean_of_differences_squared = differences_squared.mean()
    rmse_val = np.sqrt(mean_of_differences_squared)
    return rmse_val
MSEerror_DESVM= mean_squared_error(y_test, y_pred)
print('RMSEerror of DE-SVM',MSEerror_DESVM)
```

GA-SVM

```
data=y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20,
    random_state=7)
print('Training Features Shape:', X_train.shape)
print('Training Labels Shape:', Y_train .shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Labels Shape:', Y_test.shape)
Cs = [2, 3, 1.9,3, 2]
gammas = [1.9,1.7, 1.5, 3]
tol=[0.001,.01,.1,.01,2]
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf.fit(X_train, Y_train)
y_pred=clf.predict(X_test)
print(y_pred)
c=clf.score(X_test,Y_test)
from sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(Y_test,y_pred)
print ('Confusion matrix of GASVM',cm2)
print('GA-SVM graphs')
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
```

```
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.plot(Y_test,'-b',label='Test Data')
#plt.ylim(0,4)
plt.plot(y_pred,'-r',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TestData','TestNet'), frameon=True)
plt.savefig('GA-SVM_test.eps')
plt.savefig('GA-SVM_test.png')
plt.show();
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=(8,6))
plt.rc('font', size=28)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.ylabel('Amount of packet loss'); plt.xlabel('Available data');
plt.xlim(0,80)
plt.plot(Y_train ,'-g',label='TrainData')
plt.plot(y_pred,'-b',label='SVR')
legend = plt.legend(loc='upper center', shadow=True, fontsize='small',frameon=
    True)
plt.gca().legend(('TrainData','TrainNet'), frameon=True)
plt.savefig('GA-SVM_train.eps')
plt.savefig('GA-SVM_train.png')
plt.show();
b=clf.score(X_test,Y_test)
b=b*100
print ('Accuracy of GAsvm',b)
MSE=metrics.mean_squared_error(Y_test,y_pred)
def MAPE(Y_test, y_pred):                        # Formula of MAPE used
    Y_test=abs(Y_test);y_pred=abs(y_pred);
    Y_test, y_pred = np.array(Y_test), np.array(y_pred)
    return np.mean(np.abs((Y_test - y_pred) / Y_test)) * 100
from sklearn.metrics import mean_absolute_error
MapeErr2=MAPE(Y_test, y_pred)
MAEerr2=mean_absolute_error(Y_test, y_pred);
print('MAPE of GASVM',MapeErr2)
print('MAE of GASVM',MAEerr2)
print('Accuracy of GASVM',100-MapeErr2)
def rmse(y_pred, Y_test):
    differences = y_pred - Y_test                        #the DIFFERENCEs.
    differences_squared = differences ** 2                     #the SQUAREs of ^
    mean_of_differences_squared = differences_squared.mean()  #the MEAN of ^
    rmse_val = np.sqrt(mean_of_differences_squared)           #ROOT of ^
    return rmse_val
MSEerror_GASVM= mean_squared_error(Y_test, y_pred)
print('RMSEerror of GASVM',MSEerror_GASVM)
```

Grey Wolf Optimization Algorithm

```
lb=0
ub=4
dim=3
#popnum=0
#maxiers=0
Max_iter=50
```

```
SearchAgents_no=5
% initialize alpha, beta, and delta_pos
Alpha_pos=numpy.zeros(dim)
Alpha_score=float("inf")
Beta_pos=numpy.zeros(dim)
Beta_score=float("inf")
Delta_pos=numpy.zeros(dim)
Delta_score=float("inf")
%Initialize the positions of search agents
Positions=numpy.random.uniform(0,1,(SearchAgents_no,dim)) *(ub-lb)+lb
%Main loop
for l in range(0,Max_iter):
    for i in range(0,SearchAgents_no):
        % Return back the search agents that go beyond the boundaries of the
    search space
        Positions[i,:]=numpy.clip(Positions[i,:], lb, ub)
        % Calculate objective function for each search agent
        fitness=svn.svm(Positions[i,:])
        %Update Alpha, Beta, and Delta
        if fitness<Alpha_score :
            Alpha_score=fitness; # Update alpha
            Alpha_pos=Positions[i,:].copy()
        if (fitness>Alpha_score and fitness<Beta_score ):
            Beta_score=fitness  # Update beta
            Beta_pos=Positions[i,:].copy()
        if (fitness>Alpha_score and fitness>Beta_score and fitness<Delta_score):
            Delta_score=fitness # Update delta
            Delta_pos=Positions[i,:].copy()
    a=2-l*((2)/Max_iter); # a decreases linearly fron 2 to 0
    % Update the Position of search agents including omegas
    for i in range(0,SearchAgents_no):
        for j in range (0,dim):

            r1=random.random() # r1 is a random number in [0,1]
            r2=random.random() # r2 is a random number in [0,1]

            A1=2*a*r1-a; # Equation (3.3)
            C1=2*r2; # Equation (3.4)

            D_alpha=abs(C1*Alpha_pos[j]-Positions[i,j]); # Equation (3.5)-part 1
            X1=Alpha_pos[j]-A1*D_alpha; # Equation (3.6)-part 1

            r1=random.random()
            r2=random.random()

            A2=2*a*r1-a; # Equation (3.3)
            C2=2*r2; # Equation (3.4)

            D_beta=abs(C2*Beta_pos[j]-Positions[i,j]); # Equation (3.5)-part 2
            X2=Beta_pos[j]-A2*D_beta; # Equation (3.6)-part 2

            r1=random.random()
            r2=random.random()

            A3=2*a*r1-a; # Equation (3.3)
            C3=2*r2; # Equation (3.4)

            D_delta=abs(C3*Delta_pos[j]-Positions[i,j]); # Equation (3.5)-part 3
```

```
            X3=Delta_pos[j]-A3*D_delta; # Equation (3.5)-part 3

            Positions[i,j]=(X1+X2+X3)/3  # Equation (3.7)

```

Differential Evolution Algorithm

```
bounds=[(1,15),(1,20)]
mut=0.8
crossp=0.7
popsize=50
its=3
fitness=[]
dimensions = len(bounds)
pop = np.random.rand(popsize, dimensions)
min_b, max_b = np.asarray(bounds).T
diff = np.fabs(min_b - max_b)
pop_denorm = min_b + pop * diff

for x in range(0,popsize-1):

    fitness.append(svn.svm(pop_denorm[x]))
best_idx = np.argmin(fitness)
best = pop_denorm[best_idx]
for i in range(its):
    print(i)
    for j in range(popsize-1):
            idxs = [idx for idx in range(popsize) if idx != j]
            a, b, c = pop[np.random.choice(idxs, 3, replace = False)]
            mutant = np.clip(a + mut * (b - c), 0, 1)
            cross_points = np.random.rand(dimensions) < crossp
            if not np.any(cross_points):
                cross_points[np.random.randint(0, dimensions)] = True
            trial = np.where(cross_points, mutant, pop[j])
            trial_denorm = min_b + trial * diff
            f=svn.svm(trial_denorm)
            if f < fitness[j]:
                fitness[j] = f
                pop[j] = trial
                if f < fitness[best_idx]:
                    best_idx = j
                    best = trial_denorm
print(best,fitness[best_idx])
```

Genetic Algorithm

```
ga = pyeasyga.GeneticAlgorithm(data,
                                population_size=50,
                                generations=5,
                                crossover_probability=0.7,
                                mutation_probability=0.3,
                                elitism=True,
                                maximise_fitness=True)
def create_individual(data):
    return [np.random.randint(-4, 4)
for _ in range(len(Y))]
```

```
ga.create_individual = create_individual
def crossover(parent_1, parent_2):
    crossover_index = random.randrange(1, len(parent_1))
    child_1 = parent_1[:index] + parent_2[index:]
    child_2 = parent_2[:index] + parent_1[index:]
    return
    child_1, child_2
ga.crossover_function = crossover
def mutate(individual):
    mutate_index = random.randrange(len(individual))
    if individual[mutate_index] == 0:
       individual[mutate_index] == 1
    else:
       individual[mutate_index] == 0
       ga.mutate_function = mutate
   def selection(population):
    return
    random.choice(population)
      ga.selection_function = selection

def fitness (individual, data):
    fitness = 0
    if individual.count(1) == 2:
       for (selected, (X, Y)) in zip(individual, data):
        if selected:
            fitness += data
    return fitness
ga.fitness_function = fitness
ga.run()
```

Random forest

```
rf = RandomForestRegressor(n_estimators = 9, random_state = 42)
rf.fit(X_train, Y_train);
rfr=rf.predict(X_test)
MSE1=metrics.mean_squared_error(Y_test,rfr)
d=rf.score(X_test,Y_test)
print ('Accuracy of RF',d)
def MAPE(Y_test, rfr):                    # Formula of MAPE used
    Y_test=abs(Y_test);rfr=abs(rfr);
    Y_test, rfr = np.array(Y_test), np.array(rfr)
    return np.mean(np.abs((Y_test - rfr) / Y_test)) * 100
MapeErr3=MAPE(Y_test, rfr)
MAEerr3=mean_absolute_error(Y_test, rfr);
print('MAPE of RF',MapeErr3)
print('MAE of RF',MAEerr3)
print('Accuracy of RF',100-MapeErr3)
def rmse(rfr, Y_test):
    differences = rfr - Y_test                      #the DIFFERENCEs.
    differences_squared = differences ** 2                   #the SQUAREs of ^
    mean_of_differences_squared = differences_squared.mean()  #the MEAN of ^
    rmse_val = np.sqrt(mean_of_differences_squared)          #ROOT of ^
    return rmse_val
MSEerror_RF =mean_squared_error(Y_test, rfr)
print('RMSEerror of RF',MSEerror_RF)
```

Naive Bayes

```
std_scale = preprocessing.StandardScaler().fit(features[['B', 'C','R','RT']])
df_std = std_scale.transform(features[['B', 'C','R','RT']])
minmax_scale = preprocessing.MinMaxScaler().fit(features[['B', 'C','R','RT']])
df_minmax = minmax_scale.transform(features[['B', 'C','R','RT']])
Y=features['RT']
X= features.drop('RT', axis = 1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20,
    random_state=7)
model = GaussianNB().fit(X_train, Y_train)
predicted = model.predict(X_test)
ee=model.score(X_test,Y_test)
MSE2=metrics.mean_squared_error(Y_test,predicted)
def MAPE(Y_test, predicted):
    Y_test=abs(Y_test);predicted=abs(predicted);
    Y_test, predicted = np.array(Y_test), np.array(predicted)
    return np.mean(np.abs((Y_test - predicted) / Y_test)) * 100
from sklearn.metrics import mean_absolute_error
MapeErr4=MAPE(Y_test, predicted)
MAEerr4=mean_absolute_error(Y_test, predicted);
print('MAPE of NB',MapeErr4)
print('MAE of NB',MAEerr4)
print('Accuracy of NB',100-MapeErr4)
def rmse(predicted, Y_test):
    differences = predicted - Y_test
    differences_squared = differences ** 2
    mean_of_differences_squared = differences_squared.mean()
    rmse_val = np.sqrt(mean_of_differences_squared)
    return rmse_val
MSEerror_NB= mean_squared_error(Y_test, predicted)
print('RMSEerror of NB',MSEerror_NB)
```

K-Nearest Neighbour

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='
    minkowski',metric_params=None, n_jobs=2, n_neighbors=5, p=1,weights='uniform
    ')
classifier.fit(X_train, Y_train)
pred = classifier.predict(X_test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20,
    random_state=7)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
MSE3=metrics.mean_squared_error(Y_test,pred)
def MAPE(Y_test, pred):                    # Formula of MAPE used
    Y_test=abs(Y_test);pred=abs(pred);
    Y_test, pred = np.array(Y_test), np.array(pred)
    return np.mean(np.abs((Y_test - pred) / Y_test)) * 100
from sklearn.metrics import mean_absolute_error
MapeErr5=MAPE(Y_test, pred)
MAEerr5=mean_absolute_error(Y_test, pred);
print('MAPE of KNN',MapeErr5)
print('MAE of KNN',MAEerr5)
print('Accuracy of KNN',100-MapeErr5)
def rmse(pred, Y_test):
```

```
    differences = pred - Y_test
    differences_squared = differences ** 2
    mean_of_differences_squared = differences_squared.mean()
    rmse_val = np.sqrt(mean_of_differences_squared)
    return rmse_val
MSEerror_KNN= mean_squared_error(Y_test, pred)
print('RMSEerror of NB',MSEerror_KNN)
f=clf.score(X_test,Y_test)
```

Stochastic Gradient Descent

```
SGD=SGDClassifier(loss="hinge", penalty="l2",  random_state=7)
SGD.fit(X_train,Y_train);
prediction_SGD=SGD.predict(X_test)
MSE7=metrics.mean_squared_error(Y_test,prediction_SGD)
from sklearn.metrics import mean_absolute_error
MAEerr7=mean_absolute_error(Y_test, prediction_SGD);
print('MAE of RF',MAEerr7)
def rmse(prediction_SGD, Y_test):
    differences = prediction_SGD - Y_test
    differences_squared = differences ** 2
    mean_of_differences_squared = differences_squared.mean()
    rmse_val = np.sqrt(mean_of_differences_squared)
    return rmse_val
MSEerror_SGD =mean_squared_error(Y_test, prediction_SGD)
print('RMSEerror of SGD',MSEerror_SGD)
g=SGD.score(X_test,Y_test)
```

Stochastic Gradient Descent

```
%%%%%%% ------Multilayer Perceptron Classifier-----%%%%%%%%%
model1 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model1.fit(X_train, Y_train);
prediction_MLP=model1.predict(X_test)
MSE8=metrics.mean_squared_error(Y_test,prediction_MLP)
from sklearn.metrics import mean_absolute_error
MAEerr8=mean_absolute_error(Y_test, prediction_MLP);
print('MAE of RF',MAEerr8)
def rmse(prediction_MLP, Y_test):
    differences = prediction_MLP - Y_test                          #the DIFFERENCEs.
    differences_squared = differences ** 2                       #the SQUAREs of ^
    mean_of_differences_squared = differences_squared.mean()  #the MEAN of ^
    rmse_val = np.sqrt(mean_of_differences_squared)           #ROOT of ^
    return rmse_val
MSEerror_MLP =mean_squared_error(Y_test, prediction_MLP)
print('RMSEerror of MLP',MSEerror_MLP)
h=model1.score(X_test,Y_test)
```

MSE Calculation

```
%%%%%%%%--- MSE bar plot----- %%%%%%%%%%%%
fig = plt.figure(figsize=(17,8))
print('MSE of GASVM,RF,NB,KNN,GWOSVM,DE-SVM,SGD,MLP',MSE,MSE1,MSE2,MSE3,MSEGWO,
    MSE6,MSE7,MSE8)
```

```
plt.style.use('seaborn-whitegrid')
plt.rc('font', size=30)
plt.rc('xtick', labelsize=27)
plt.rc('ytick', labelsize=27)
width=1/2
plt.ylabel('Mean Square Error')
plt.bar(['GASVM','RF','NB','KNN','GWOSVM','DESVM','SGD','MLP'],[MSE,MSE1,MSE2,
    MSE3,MSEGWO,MSE6,MSE7,MSE8], width,align='center', alpha=0.9)
plt.savefig('MSE.eps')
plt.savefig('MSE.png')
plt.show()
```

MAE Calculation

```
%%%%%%% ------ MAE bar plot----------%%%%%%%%%%%
fig = plt.figure(figsize=(17,8))
print('MAE of GASVM,RF,NB,KNN,GWOSVM,DESVM,SGD,MLP',MAEerr2,MAEerr3,MAEerr4,
    MAEerr5,MAEerr1, MAEerr6,MAEerr7,MAEerr8)
width=1/2
plt.style.use('seaborn-whitegrid')
plt.rc('font', size=30)
plt.rc('xtick', labelsize=27)
plt.rc('ytick', labelsize=27)
plt.ylabel('Mean Absolute Error')
plt.bar(['GASVM','RF','NB','KNN','GWOSVM','DESVM','SGD','MLP'],[MAEerr2,MAEerr3,
    MAEerr4,MAEerr5,MAEerr1,MAEerr6,MAEerr7,MAEerr8], width,align='center', alpha
    =.9, color = 'b')
plt.savefig('MAE.eps')
plt.savefig('MAE.png')
plt.show()
```

RMSE Calculation

```
%%%%%%% ------ RMSE bar plot----------%%%%%%%%%%%
fig = plt.figure(figsize=(17,8))
print('RMSE of GASVM,RF,NB,KNN,GWOSVM,DE-SVM, SGD,MLP',MSEerror_GASVM,MSEerror_RF
    ,MSEerror_NB,MSEerror_KNN,MSEerror_GWOSVM, MSEerror_DESVM,MSEerror_SGD,
    MSEerror_MLP )
width=1/2
plt.style.use('seaborn-whitegrid')
plt.rc('font', size=30)
plt.rc('xtick', labelsize=27)
plt.rc('ytick', labelsize=27)
plt.ylabel('Root Mean Square Error')
plt.bar(['GASVM','RF','NB','KNN','GWOSVM','DESVM','SGD','MLP'],[MSEerror_GASVM,
    MSEerror_RF,MSEerror_NB,MSEerror_KNN,MSEerror_GWOSVM,MSEerror_DESVM,
    MSEerror_SGD,MSEerror_MLP ], width,align='center', alpha=.99, color = '0.2')
plt.savefig('RMSE.eps')
plt.savefig('RMSE.png')
plt.show()
```

Fault Detection

```
%%%%%% Fault Detection %%%%%%%%%%%
%%% Import feature from GAIN FAULT datasets %%%
```

Prepared Datasets

```
feature1 = pd.read_csv('Gain_10.csv')
feature1.describe()
feature2 = pd.read_csv('Gain_20.csv')
feature2.describe()
feature3 = pd.read_csv('Gain_30.csv')
feature3.describe()
feature4 = pd.read_csv('Gain_40.csv')
feature4.describe()
feature5 = pd.read_csv('Gain_50.csv')
feature5.describe()
%%%  Import features from OFFSET FAULT datasets%%%
feature11 = pd.read_csv('offset_10.csv')
feature11.describe()
feature12 = pd.read_csv('offset_20.csv')
feature12.describe()
feature13 = pd.read_csv('offset_30.csv')
feature13.describe()
feature14 = pd.read_csv('offset_40.csv')
feature14.describe()
feature15 = pd.read_csv('offset_50.csv')
feature15.describe()
%%%  Import features from STUCKAT FAULT datasets%%%
feature16 = pd.read_csv('stuck_10.csv')
feature16.describe()
feature17 = pd.read_csv('stuck_20.csv')
feature17.describe()
feature18 = pd.read_csv('stuck_30.csv')
feature18.describe()
feature19 = pd.read_csv('stuck_40.csv')
feature19.describe()
feature20 = pd.read_csv('stuck_50.csv')
feature20.describe()
%%% Import features from OUT OF BOUND FAULT datasets %%%%
feature161 = pd.read_csv('OB_10.csv')
feature161.describe()
feature171 = pd.read_csv('OB_20.csv')
feature171.describe()
feature181 = pd.read_csv('OB_30.csv')
feature181.describe()
feature191 = pd.read_csv('OB_40.csv')
feature191.describe()
feature201 = pd.read_csv('OB_50.csv')
feature201.describe()
X1=np.array(feature1.drop('y',axis=1))
Y1=np.array(feature1['y'])
X2=np.array(feature2.drop('y',axis=1))
Y2=np.array(feature2['y'])
X3=np.array(feature3.drop('y',axis=1))
Y3=np.array(feature3['y'])
X4=np.array(feature4.drop('y',axis=1))
Y4=np.array(feature4['y'])
X5=np.array(feature5.drop('y',axis=1))
Y5=np.array(feature5['y'])
X11=np.array(feature11.drop('y',axis=1))
Y11=np.array(feature11['y'])
X12=np.array(feature12.drop('y',axis=1))
```

```
Y12=np.array(feature12['y'])
X13=np.array(feature13.drop('y',axis=1))
Y13=np.array(feature13['y'])
X14=np.array(feature14.drop('y',axis=1))
Y14=np.array(feature14['y'])
X15=np.array(feature15.drop('y',axis=1))
Y15=np.array(feature15['y'])
X16=np.array(feature16.drop('y',axis=1))
Y16=np.array(feature16['y'])
X17=np.array(feature17.drop('y',axis=1))
Y17=np.array(feature17['y'])
X18=np.array(feature18.drop('y',axis=1))
Y18=np.array(feature18['y'])
X19=np.array(feature19.drop('y',axis=1))
Y19=np.array(feature19['y'])
X20=np.array(feature20.drop('y',axis=1))
Y20=np.array(feature20['y'])
X161=np.array(feature161.drop('y',axis=1))
Y161=np.array(feature161['y'])
X171=np.array(feature171.drop('y',axis=1))
Y171=np.array(feature171['y'])
X181=np.array(feature181.drop('y',axis=1))
Y181=np.array(feature181['y'])
X191=np.array(feature191.drop('y',axis=1))
Y191=np.array(feature191['y'])
X201=np.array(feature201.drop('y',axis=1))
Y201=np.array(feature201['y'])
%%%% Divide datasets for training and testing
%%%% GAIN FAULT %%%%%
X1_train, X1_test, Y1_train, Y1_test = cross_validation.train_test_split(X1, Y1,
    test_size=0.66, random_state=7)
print('Training Features Shape:', X1_train.shape)
print('Training Labels Shape:', Y1_train.shape)
print('Testing Features Shape:', X1_test.shape)
print('Testing Labels Shape:', Y1_test.shape)
X2_train, X2_test, Y2_train, Y2_test = cross_validation.train_test_split(X2, Y2,
    test_size=0.66, random_state=7)
print('Training Features Shape:', X2_train.shape)
print('Training Labels Shape:', Y2_train.shape)
print('Testing Features Shape:', X2_test.shape)
print('Testing Labels Shape:', Y2_test.shape)
X3_train, X3_test, Y3_train, Y3_test = cross_validation.train_test_split(X3, Y3,
    test_size=0.66, random_state=7)
print('Training Features Shape:', X3_train.shape)
print('Training Labels Shape:', Y3_train.shape)
print('Testing Features Shape:', X3_test.shape)
print('Testing Labels Shape:', Y3_test.shape)
X4_train, X4_test, Y4_train, Y4_test = cross_validation.train_test_split(X4, Y4,
    test_size=0.66, random_state=7)
print('Training Features Shape:', X4_train.shape)
print('Training Labels Shape:', Y4_train.shape)
print('Testing Features Shape:', X4_test.shape)
print('Testing Labels Shape:', Y4_test.shape)
X5_train, X5_test, Y5_train, Y5_test = cross_validation.train_test_split(X5, Y5,
    test_size=0.66, random_state=7)
print('Training Features Shape:', X5_train.shape)
print('Training Labels Shape:', Y5_train.shape)
print('Testing Features Shape:', X5_test.shape)
```

```
print('Testing Labels Shape:', Y5_test.shape)
%%%%% OUT OF BOUND FAULT %%%%%%%
X11_train, X11_test, Y11_train, Y11_test = cross_validation.train_test_split(X11,
     Y11, test_size=0.66, random_state=7)
print('Training Features Shape:', X11_train.shape)
print('Training Labels Shape:', Y11_train.shape)
print('Testing Features Shape:', X11_test.shape)
print('Testing Labels Shape:', Y11_test.shape)
X12_train, X12_test, Y12_train, Y12_test = cross_validation.train_test_split(X12,
     Y12, test_size=0.66, random_state=7)
print('Training Features Shape:', X12_train.shape)
print('Training Labels Shape:', Y12_train.shape)
print('Testing Features Shape:', X12_test.shape)
print('Testing Labels Shape:', Y12_test.shape)
X13_train, X13_test, Y13_train, Y13_test = cross_validation.train_test_split(X13,
     Y13, test_size=0.66, random_state=7)
print('Training Features Shape:', X13_train.shape)
print('Training Labels Shape:', Y13_train.shape)
print('Testing Features Shape:', X13_test.shape)
print('Testing Labels Shape:', Y13_test.shape)
X14_train, X14_test, Y14_train, Y14_test = cross_validation.train_test_split(X14,
     Y14, test_size=0.66, random_state=7)
print('Training Features Shape:', X14_train.shape)
print('Training Labels Shape:', Y14_train.shape)
print('Testing Features Shape:', X14_test.shape)
print('Testing Labels Shape:', Y14_test.shape)
X15_train, X15_test, Y15_train, Y15_test = cross_validation.train_test_split(X15,
     Y15, test_size=0.66, random_state=7)
print('Training Features Shape:', X15_train.shape)
print('Training Labels Shape:', Y15_train.shape)
print('Testing Features Shape:', X15_test.shape)
print('Testing Labels Shape:', Y15_test.shape)
%%%%% STUCKAT FAULT %%%%%%%
X16_train, X16_test, Y16_train, Y16_test = cross_validation.train_test_split(X16,
     Y16, test_size=0.66, random_state=7)
print('Training Features Shape:', X16_train.shape)
print('Training Labels Shape:', Y16_train.shape)
print('Testing Features Shape:', X16_test.shape)
print('Testing Labels Shape:', Y16_test.shape)
X17_train, X17_test, Y17_train, Y17_test = cross_validation.train_test_split(X17,
     Y17, test_size=0.66, random_state=7)
print('Training Features Shape:', X17_train.shape)
print('Training Labels Shape:', Y17_train.shape)
print('Testing Features Shape:', X17_test.shape)
print('Testing Labels Shape:', Y17_test.shape)
X18_train, X18_test, Y18_train, Y18_test = cross_validation.train_test_split(X18,
     Y18, test_size=0.66, random_state=7)
print('Training Features Shape:', X18_train.shape)
print('Training Labels Shape:', Y18_train.shape)
print('Testing Features Shape:', X18_test.shape)
print('Testing Labels Shape:', Y18_test.shape)
X19_train, X19_test, Y19_train, Y19_test = cross_validation.train_test_split(X19,
     Y19, test_size=0.66, random_state=7)
print('Training Features Shape:', X19_train.shape)
print('Training Labels Shape:', Y19_train.shape)
print('Testing Features Shape:', X19_test.shape)
print('Testing Labels Shape:', Y19_test.shape)
```

```
X20_train, X20_test, Y20_train, Y20_test = cross_validation.train_test_split(X20,
     Y20, test_size=0.66, random_state=7)
print('Training Features Shape:', X20_train.shape)
print('Training Labels Shape:', Y20_train.shape)
print('Testing Features Shape:', X20_test.shape)
print('Testing Labels Shape:', Y20_test.shape)
Y=Y1_train
%%%% OFFSET FAULT %%%%%%%
X161_train, X161_test, Y161_train, Y161_test = cross_validation.train_test_split(
    X161, Y161, test_size=0.66, random_state=7)
print('Training Features Shape:', X161_train.shape)
print('Training Labels Shape:', Y161_train.shape)
print('Testing Features Shape:', X161_test.shape)
print('Testing Labels Shape:', Y161_test.shape)
X171_train, X171_test, Y171_train, Y171_test = cross_validation.train_test_split(
    X171, Y171, test_size=0.66, random_state=7)
print('Training Features Shape:', X171_train.shape)
print('Training Labels Shape:', Y171_train.shape)
print('Testing Features Shape:', X171_test.shape)
print('Testing Labels Shape:', Y171_test.shape)
X181_train, X181_test, Y181_train, Y181_test = cross_validation.train_test_split(
    X181, Y181, test_size=0.66, random_state=7)
print('Training Features Shape:', X181_train.shape)
print('Training Labels Shape:', Y181_train.shape)
print('Testing Features Shape:', X181_test.shape)
print('Testing Labels Shape:', Y181_test.shape)
X191_train, X191_test, Y191_train, Y191_test = cross_validation.train_test_split(
    X191, Y191, test_size=0.66, random_state=7)
print('Training Features Shape:', X191_train.shape)
print('Training Labels Shape:', Y191_train.shape)
print('Testing Features Shape:', X191_test.shape)
print('Testing Labels Shape:', Y191_test.shape)
X201_train, X201_test, Y201_train, Y201_test = cross_validation.train_test_split(
    X201, Y201, test_size=0.66, random_state=7)
print('Training Features Shape:', X201_train.shape)
print('Training Labels Shape:', Y201_train.shape)
print('Testing Features Shape:', X201_test.shape)
print('Testing Labels Shape:', Y201_test.shape)
```

GWO-SVM

```
%%%%%%%% Apply Classifiers for detection of GAIN FAULT in dataset 1 %%%%%%%%
%%% GWO-SVM %%%%
Cs = [1, 1.99, 1.1,1.89, 1]
gammas = [1.9,1.7, 1.5, 3,1]
tol=[0.001,.01,.1,.01,2]
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf1 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf1.fit(X1_train, Y1_train)
prediction1=clf1.predict(X1_test)
accuracy_gain_1=clf1.score(X1_test, Y1_test)
print('Accuracy GWOSVM_gain1:',accuracy_gain_1)
cm=confusion_matrix(Y1_test, prediction1)
print(cm)
MCC_1_SVM = matthews_corrcoef(Y1_test, prediction1)
print ('MCC_1_GWOSVM', MCC_1_SVM)
```

Random Forest

```
rfc = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc.fit(X1_train, Y1_train);
accuracy_gain_RFC_1=rfc.score(X1_test, Y1_test)
print('Accuracy RF_gain1:',accuracy_gain_RFC_1)
prediction_RFC=rfc.predict(X1_test)
MCC_1_RFC = matthews_corrcoef(Y1_test, prediction_RFC)
print ('MCC_1_RFC', MCC_1_RFC)
```

Stochastic Gradient Descent

```
SGD=SGDClassifier(loss="hinge", penalty="l2",  random_state=7)
SGD.fit(X1_train,Y1_train);
accuracy_gain_SGD_1=SGD.score(X1_test, Y1_test);
print('Accuracy SGD_gain1:',accuracy_gain_SGD_1)
prediction_SGD=SGD.predict(X1_test)
MCC_1_SGD = matthews_corrcoef(Y1_test, prediction_SGD)
print ('MCC_1_SGD', MCC_1_SGD)
```

Multilayer Perceptron

```
model1 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model1.fit(X1_train, Y1_train);
accuracy_gain_MLP_1=model1.score(X1_test,Y1_test)
print('Accuracy_MLP_gain1:',accuracy_gain_MLP_1)
prediction_MLP=model1.predict(X1_test)
MCC_1_MLP = matthews_corrcoef(Y1_test, prediction_MLP)
print ('MCC_1_MLP', MCC_1_MLP)
%%%%%%%% Apply classifiers for detection of GAIN FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf2 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf2.fit(X2_train, Y2_train)
accuracy_gain_2=clf2.score(X2_test, Y2_test)
print('Accuracy GWOSVMM_gain2:',accuracy_gain_2)
rfc2 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc2.fit(X2_train, Y2_train);
accuracy_gain_RFC_2=rfc2.score(X2_test, Y2_test)
print('Accuracy_RFC_gain2:',accuracy_gain_RFC_2)
SGD2=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD2.fit(X2_train,Y2_train);
accuracy_gain_SGD_2=SGD2.score(X2_test, Y2_test);
print('Accuracy_SGD_gain2:',accuracy_gain_SGD_2)
model2 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model2.fit(X2_train, Y2_train);
accuracy_gain_MLP_2=model2.score(X2_test,Y2_test)
print('Accuracy_MLP_gain2:',accuracy_gain_MLP_2)
prediction_MLP=model1.predict(X1_test)
MCC_1_MLP = matthews_corrcoef(Y1_test, prediction_MLP)
print ('MCC_1_MLP', MCC_1_MLP)
%%%%%%%% Apply classifiers for detection of GAIN FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf3 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
```

```
clf3.fit(X3_train, Y3_train)
accuracy_gain_3=clf3.score(X3_test, Y3_test)
print('Accuracy GWOSVM_gain3:',accuracy_gain_3)
rfc3 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc3.fit(X3_train, Y3_train);
accuracy_gain_RFC_3=rfc3.score(X3_test, Y3_test)
print('Accuracy RF_gain3:',accuracy_gain_RFC_3)
SGD3=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD3.fit(X3_train,Y3_train);
accuracy_gain_SGD_3=SGD3.score(X3_test, Y3_test);
print('Accuracy_SGD gain3:',accuracy_gain_SGD_3)
model3 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model3.fit(X3_train, Y3_train);
accuracy_gain_MLP_3=model3.score(X3_test,Y3_test)
print('Accuracy_MLP gain3:',accuracy_gain_MLP_3)
prediction_MLP=model1.predict(X1_test)
MCC_1_MLP = matthews_corrcoef(Y1_test, prediction_MLP)
print ('MCC_1_MLP', MCC_1_MLP)
%%%%%%%% Apply classifiers for detection of GAIN FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf4 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf4.fit(X4_train, Y4_train)
accuracy_gain_4=clf4.score(X4_test, Y4_test)
print('Accuracy GWOSVM_gain4:',accuracy_gain_4)
rfc4 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc4.fit(X4_train, Y4_train);
accuracy_gain_RFC_4=rfc4.score(X4_test, Y4_test)
print('Accuracy RF_gain4:',accuracy_gain_RFC_4)
SGD4=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD4.fit(X4_train,Y4_train);
accuracy_gain_SGD_4=SGD4.score(X4_test, Y4_test);
print('Accuracy_SGD gain3:',accuracy_gain_SGD_4)
model4 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model4.fit(X4_train, Y4_train);
accuracy_gain_MLP_4=model4.score(X4_test,Y4_test)
print('Accuracy_MLP gain4:',accuracy_gain_MLP_4)
prediction_MLP=model1.predict(X1_test)
MCC_1_MLP = matthews_corrcoef(Y1_test, prediction_MLP)
print ('MCC_1_MLP', MCC_1_MLP)
%%%%%%%% Apply classifiers for detection of GAIN FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf5 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf5.fit(X5_train, Y5_train)
accuracy_gain_5=clf5.score(X5_test, Y5_test)
print('Accuracy GWOSVM gain5:',accuracy_gain_5)
prediction_MLP_clf5=clf5.predict(X1_test)
MCC_1_MLP_5_clf = matthews_corrcoef(Y1_test, prediction_MLP_clf5)
print ('MCC_1_MLP_5', MCC_1_MLP_5_clf)
rfc5 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc5.fit(X5_train, Y5_train);
accuracy_gain_RFC_5=rfc5.score(X5_test, Y5_test)
print('Accuracy RF_gain5:',accuracy_gain_RFC_5)
prediction_MLP_rfc5=rfc5.predict(X1_test)
MCC_MLP_5_rfc = matthews_corrcoef(Y1_test, prediction_MLP_rfc5)
```

```
print ('MCC_1_MLP_5', MCC_MLP_5_rfc)
SGD5=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD5.fit(X5_train,Y5_train);
accuracy_gain_SGD_5=SGD5.score(X5_test, Y5_test);
print('Accuracy_SGD gain5:',accuracy_gain_SGD_5)
prediction_MLP_SGD5=SGD5.predict(X1_test)
MCC_MLP_5_SGD = matthews_corrcoef(Y1_test, prediction_MLP_SGD5)
print ('MCC_1_MLP_5', MCC_MLP_5_SGD)
model5 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model5.fit(X5_train, Y5_train);
accuracy_gain_MLP_5=model5.score(X5_test,Y5_test)
print('Accuracy_MLP gain5:',accuracy_gain_MLP_5)
prediction_MLP_5=model5.predict(X1_test)
MCC_1_MLP_5 = matthews_corrcoef(Y1_test, prediction_MLP_5)
print ('MCC_1_MLP_5', MCC_1_MLP_5)
print('gain mcc:', MCC_1_MLP_5, MCC_MLP_5_SGD, MCC_MLP_5_rfc, MCC_1_MLP_5_clf )
GAIN_MCC=[MCC_1_MLP_5, MCC_MLP_5_SGD, MCC_MLP_5_rfc, MCC_1_MLP_5_clf]
%%%%%%%% Apply classifiers for detection of OUT OF BOUND FAULT in dataset 1
    %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf11 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf11.fit(X11_train, Y11_train)
accuracy_OFB_11=clf11.score(X11_test, Y11_test)
print('Accuracy GWOSVM_OB1:',accuracy_OFB_11)
rfc11 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc11.fit(X11_train, Y11_train);
accuracy_OFB_RFC_11=rfc11.score(X11_test, Y11_test)
print('Accuracy RFC_OB1:',accuracy_OFB_RFC_11)
SGD11=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD11.fit(X11_train,Y11_train);
accuracy_OFB_SGD_11=SGD11.score(X11_test, Y11_test);
print('Accuracy_SGD_OB1:',accuracy_OFB_SGD_11)
model11 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model11.fit(X11_train, Y11_train);
accuracy_OFB_MLP_11=model11.score(X11_test,Y11_test)
print('Accuracy_MLP_OB1:',accuracy_OFB_MLP_11)
%%%%%%%% Apply classifiers for detection of OUT OF BOUND FAULT in dataset 2
    %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf12 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf12.fit(X12_train, Y12_train)
accuracy_OFB_12=clf12.score(X12_test, Y12_test)
print('Accuracy GWOSVM_OB2:',accuracy_OFB_12)
rfc12 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc12.fit(X12_train, Y12_train);
accuracy_OFB_RFC_12=rfc12.score(X12_test, Y12_test)
print('Accuracy RFC_OB2:',accuracy_OFB_RFC_12)
SGD12=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD12.fit(X12_train,Y12_train);
accuracy_OFB_SGD_12=SGD12.score(X12_test, Y12_test);
print('Accuracy_SGD_OB2:',accuracy_OFB_SGD_12)
model12 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
```

```
model12.fit(X12_train, Y12_train);
accuracy_OFB_MLP_12=model12.score(X12_test,Y12_test)
print('Accuracy_MLP_OB2:',accuracy_OFB_MLP_12)
%%%%%%%% Apply classifiers for detection of OUT OF BOUND FAULT in dataset 3
    %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf13 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf13.fit(X13_train, Y13_train)
accuracy_OFB_13=clf13.score(X13_test, Y13_test)
print('Accuracy GWOSVM_OB3:',accuracy_OFB_13)
rfc13 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc13.fit(X13_train, Y13_train);
accuracy_OFB_RFC_13=rfc13.score(X13_test, Y13_test)
print('Accuracy RFC_OB3:',accuracy_OFB_RFC_13)
SGD13=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD13.fit(X13_train,Y13_train);
accuracy_OFB_SGD_13=SGD13.score(X13_test, Y13_test);
print('Accuracy_SGD_OB3:',accuracy_OFB_SGD_13)
model13 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model13.fit(X13_train, Y13_train);
accuracy_OFB_MLP_13=model13.score(X13_test,Y13_test)
print('Accuracy_MLP_OB3:',accuracy_OFB_MLP_13)
%%%%%%%% Apply classifiers for detection of OUT OF BOUND FAULT in dataset 4
    %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf14 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf14.fit(X14_train, Y14_train)
accuracy_OFB_14=clf14.score(X14_test, Y14_test)
print('Accuracy GWOSVM_OB4:',accuracy_OFB_14)
rfc14 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc14.fit(X14_train, Y14_train);
accuracy_OFB_RFC_14=rfc14.score(X14_test, Y14_test)
print('Accuracy RFC OB4:',accuracy_OFB_RFC_14)
SGD14=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD14.fit(X14_train,Y14_train);
accuracy_OFB_SGD_14=SGD14.score(X14_test, Y14_test);
print('Accuracy_SGD OB4:',accuracy_OFB_SGD_14)
model14 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model14.fit(X14_train, Y14_train);
accuracy_OFB_MLP_14=model14.score(X14_test,Y14_test)
print('Accuracy_MLP OB4:',accuracy_OFB_MLP_14)
%%%%%%%% Apply classifiers for detection of OUT OF BOUND FAULT in dataset 5
    %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf15 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf15.fit(X15_train, Y15_train)
accuracy_OFB_15=clf15.score(X15_test, Y15_test)
print('Accuracy GWOSVM OB5:',accuracy_OFB_15)
prediction_MLP_clf15=clf15.predict(X1_test)
MCC_MLP_15_clf = matthews_corrcoef(Y1_test, prediction_MLP_clf15)
print ('MCC_MLP_15_clf', MCC_MLP_15_clf)
rfc15 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc15.fit(X15_train, Y15_train);
accuracy_OFB_RFC_15=rfc15.score(X15_test, Y15_test)
```

```
print('Accuracy RFC OB5:',accuracy_OFB_RFC_15)
prediction_MLP_rfc15=rfc15.predict(X1_test)
MCC_MLP_15_rfc = matthews_corrcoef(Y1_test, prediction_MLP_rfc15)
print ('MCC_MLP_15_rfc', MCC_MLP_15_rfc)
SGD15=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD15.fit(X15_train,Y15_train);
accuracy_OFB_SGD_15=SGD15.score(X15_test, Y15_test);
print('Accuracy_SGD OB5:',accuracy_OFB_SGD_15)
prediction_MLP_SGD15=SGD15.predict(X1_test)
MCC_MLP_SGD15 = matthews_corrcoef(Y1_test, prediction_MLP_SGD15)
print ('MCC_MLP_SGD15', MCC_MLP_SGD15)
model15 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model15.fit(X15_train, Y15_train);
accuracy_OFB_MLP_15=model15.score(X15_test,Y15_test)
print('Accuracy_MLP OB5:',accuracy_OFB_MLP_15)
prediction_MLP_model15=model15.predict(X1_test)
MCC_MLP_model15 = matthews_corrcoef(Y1_test, prediction_MLP_model15)
print ('MCC_MLP_model15', MCC_MLP_model15)
print('OutOfBounds mcc:',MCC_MLP_15_clf, MCC_MLP_15_rfc, MCC_MLP_SGD15,
    MCC_MLP_model15)
OFB_MCC=[MCC_MLP_15_clf, MCC_MLP_15_rfc, MCC_MLP_SGD15, MCC_MLP_model15]
%%%%%%%% Apply classifiers for detection of STUCKAT FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf16 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf16.fit(X16_train, Y16_train)
accuracy_SAT_16=clf16.score(X16_test, Y16_test)
print('Accuracy GWOSVM stuck1:',accuracy_SAT_16)
rfc16 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc16.fit(X16_train, Y16_train);
accuracy_SAT_RFC_16=rfc16.score(X16_test, Y16_test)
print('Accuracy RFC stuck1:',accuracy_SAT_RFC_16)
SGD16=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD16.fit(X16_train,Y16_train);
accuracy_SAT_SGD_16=SGD16.score(X16_test, Y16_test);
print('Accuracy_SGD stuck1:',accuracy_SAT_SGD_16)
model16 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model16.fit(X16_train, Y16_train);
accuracy_SAT_MLP_16=model16.score(X16_test,Y16_test)
print('Accuracy_MLP stuck1:',accuracy_SAT_MLP_16)
%%%%%%%% Apply classifiers for detection of STUCKAT FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf17 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf17.fit(X17_train, Y17_train)
accuracy_SAT_17=clf17.score(X17_test, Y17_test)
print('Accuracy GWOSVM stuck2:',accuracy_SAT_17)
rfc17 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc17.fit(X17_train, Y17_train);
accuracy_SAT_RFC_17=rfc17.score(X17_test, Y17_test)
print('Accuracy RFC stuck2:',accuracy_SAT_RFC_17)
SGD17=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD17.fit(X17_train,Y17_train);
accuracy_SAT_SGD_17=SGD17.score(X17_test, Y17_test);
print('Accuracy_SGD stuck2:',accuracy_SAT_SGD_17)
```

```
model17 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model17.fit(X17_train, Y17_train);
accuracy_SAT_MLP_17=model17.score(X17_test,Y17_test)
print('Accuracy_MLP stuck2:',accuracy_SAT_MLP_17)
%%%%%%%% Apply classifiers for detection of STUCKAT FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf18 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf18.fit(X18_train, Y18_train)
accuracy_SAT_18=clf18.score(X18_test, Y18_test)
print('Accuracy GWOSVM stuck3:',accuracy_SAT_18)
rfc18 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc18.fit(X18_train, Y18_train);
accuracy_SAT_RFC_18=rfc18.score(X18_test, Y18_test)
print('Accuracy RFC stuck3:',accuracy_SAT_RFC_18)
SGD18=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD18.fit(X18_train,Y18_train);
accuracy_SAT_SGD_18=SGD18.score(X18_test, Y18_test);
print('Accuracy_SGD stuck3:',accuracy_SAT_SGD_18)
model18 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model18.fit(X18_train, Y18_train);
accuracy_SAT_MLP_18=model18.score(X18_test,Y18_test)
print('Accuracy_MLP stuck3:',accuracy_SAT_MLP_18)
%%%%%%%% Apply classifiers for detection of STUCKAT FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf19 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf19.fit(X19_train, Y19_train)
accuracy_SAT_19=clf19.score(X19_test, Y19_test)
print('Accuracy GWOSVMstuck4:',accuracy_SAT_19)
rfc19 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc19.fit(X19_train, Y19_train);
accuracy_SAT_RFC_19=rfc19.score(X19_test, Y19_test)
print('Accuracy RFC stuck4:',accuracy_SAT_RFC_19)
SGD19=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD19.fit(X19_train,Y19_train);
accuracy_SAT_SGD_19=SGD19.score(X19_test, Y19_test);
print('Accuracy_SGD stuck4:',accuracy_SAT_SGD_19)
model19 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model19.fit(X19_train, Y19_train);
accuracy_SAT_MLP_19=model19.score(X19_test,Y19_test)
print('Accuracy_MLP stuck4:',accuracy_SAT_MLP_19)
%%%%%%%% Apply classifiers for detection of STUCKAT FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf20 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf20.fit(X20_train, Y20_train)
accuracy_SAT_20=clf20.score(X20_test, Y20_test)
print('Accuracy GWOSVM_stuck5:',accuracy_SAT_20)
prediction_clf20=clf20.predict(X1_test)
MCC_clf20 = matthews_corrcoef(Y1_test, prediction_clf20)
print ('MCC_1_clf20', MCC_clf20)
rfc20 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc20.fit(X20_train, Y20_train);
accuracy_SAT_RFC_20=rfc20.score(X20_test, Y20_test)
```

```
print('Accuracy RFC stuck5:',accuracy_SAT_RFC_20)
prediction_rfc20=rfc20.predict(X1_test)
MCC_rfc20 = matthews_corrcoef(Y1_test, prediction_rfc20)
print ('MCC_rfc20', MCC_rfc20)
SGD20=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD20.fit(X20_train,Y20_train);
accuracy_SAT_SGD_20=SGD20.score(X20_test, Y20_test);
print('Accuracy_SGD stuck5:',accuracy_SAT_SGD_20)
prediction_SGD20=SGD20.predict(X1_test)
MCC_SGD20 = matthews_corrcoef(Y1_test, prediction_SGD20)
print ('MCC_1_MLP_5 stuck5', MCC_SGD20)
model20 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model20.fit(X20_train, Y20_train);
accuracy_SAT_MLP_20=model15.score(X20_test,Y20_test)
print('Accuracy_MLP stuck5:',accuracy_SAT_MLP_20)
prediction_MLP_model20=model20.predict(X1_test)
MCC_MLP_model20 = matthews_corrcoef(Y1_test, prediction_MLP_model20)
print ('MCC_1_MLP_5', MCC_MLP_model20)
print('stuck_at_50:',MCC_clf20, MCC_rfc20 , MCC_SGD20, MCC_MLP_model20)
SAT_MCC=[MCC_clf20, MCC_rfc20 , MCC_SGD20, MCC_MLP_model20]
%%%%%%%% Apply classifiers for detection of OFFSET FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf161 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf161.fit(X161_train, Y161_train)
accuracy_OS_161=clf161.score(X161_test, Y161_test)
print('Accuracy GWOSVM_offset1:',accuracy_OS_161)
rfc161 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc161.fit(X161_train, Y161_train);
accuracy_OS_RFC_161=rfc161.score(X161_test, Y161_test)
print('Accuracy RFC offset1:',accuracy_OS_RFC_161)
SGD161=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD161.fit(X161_train,Y161_train);
accuracy_OS_SGD_161=SGD161.score(X161_test, Y161_test);
print('Accuracy_SGD offset1:',accuracy_OS_SGD_161)
model161 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model161.fit(X161_train, Y161_train);
accuracy_OS_MLP_161=model161.score(X161_test,Y161_test)
print('Accuracy_MLP offset1:',accuracy_OS_MLP_161)
%%%%%%%% Apply classifiers for detection of OFFSET FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf171 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf171.fit(X171_train, Y171_train)
accuracy_OS_171=clf171.score(X171_test, Y171_test)
print('Accuracy GWOSVM_offset2:',accuracy_OS_171)
rfc171 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc171.fit(X171_train, Y171_train);
accuracy_OS_RFC_171=rfc171.score(X171_test, Y171_test)
print('Accuracy RFC offset2:',accuracy_OS_RFC_171)
SGD171=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD171.fit(X171_train,Y171_train);
accuracy_OS_SGD_171=SGD.score(X171_test, Y171_test);
print('Accuracy_SGD offset2:',accuracy_OS_SGD_171)
```

```
model171 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model171.fit(X171_train, Y171_train);
accuracy_OS_MLP_171=model171.score(X171_test,Y171_test)
print('Accuracy_MLP:',accuracy_OS_MLP_171)
%%%%%%%% Apply classifiers for detection of OFFSET FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf181 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf181.fit(X181_train, Y181_train)
accuracy_OS_181=clf181.score(X181_test, Y181_test)
print('Accuracy GWOSVM offset3:',accuracy_OS_181)
rfc181 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc181.fit(X181_train, Y181_train);
accuracy_OS_RFC_181=rfc181.score(X181_test, Y181_test)
print('Accuracy RFC offset:',accuracy_OS_RFC_181)
SGD181=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD181.fit(X181_train,Y181_train);
accuracy_OS_SGD_181=SGD181.score(X181_test, Y181_test);
print('Accuracy_SGD offset3:',accuracy_OS_SGD_181)
model181 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model181.fit(X181_train, Y181_train);
accuracy_OS_MLP_181=model181.score(X181_test,Y181_test)
print('Accuracy_MLP:',accuracy_OS_MLP_181)
%%%%%%%% Apply classifiers for detection of OFFSET FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf191 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf191.fit(X191_train, Y191_train)
accuracy_OS_191=clf191.score(X191_test, Y191_test)
print('Accuracy GWOSVM offset4:',accuracy_OS_191)
rfc191 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc191.fit(X191_train, Y191_train);
accuracy_OS_RFC_191=rfc191.score(X191_test, Y191_test)
print('Accuracy RFC offset4:',accuracy_OS_RFC_191)
SGD191=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD191.fit(X191_train,Y191_train);
accuracy_OS_SGD_191=SGD191.score(X191_test, Y191_test);
print('Accuracy_SGD offset4:',accuracy_OS_SGD_191)
model191 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model191.fit(X191_train, Y191_train);
accuracy_OS_MLP_191=model191.score(X191_test,Y191_test)
print('Accuracy_MLP:',accuracy_OS_MLP_191)
%%%%%%%% Apply classifiers for detection of OFFSET FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol':tol}
clf201 = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=KFold(2))
clf201.fit(X201_train, Y201_train)
accuracy_OS_201=clf201.score(X201_test, Y201_test)
print('Accuracy GWOSVM offset4:',accuracy_OS_201)
prediction_MLP_clf201=clf201.predict(X1_test)
MCC_clf201 = matthews_corrcoef(Y1_test, prediction_MLP_clf201)
print ('MCC_MLP_clf201', MCC_clf201)
rfc201 = RandomForestClassifier(n_estimators = 9, random_state = 42)
rfc201.fit(X201_train, Y201_train);
accuracy_OS_RFC_201=rfc201.score(X201_test, Y201_test)
```

```
print('Accuracy RFC offset4:',accuracy_OS_RFC_201)
prediction_rfc201=rfc201.predict(X1_test)
MCC_rfc201 = matthews_corrcoef(Y1_test, prediction_rfc201)
print ('MCC_RFC201', MCC_rfc201)
SGD201=SGDClassifier(loss="hinge", penalty="l2", random_state=7)
SGD201.fit(X201_train,Y201_train);
accuracy_OS_SGD_201=SGD201.score(X201_test, Y201_test);
print('Accuracy_SGD offset4:',accuracy_OS_SGD_201)
prediction_MLP_SGD201=SGD201.predict(X1_test)
MCC_MLP_SGD201 = matthews_corrcoef(Y1_test, prediction_MLP_SGD201)
print ('MCC_SGD201', MCC_MLP_SGD201)
model201 = MLPClassifier(alpha=0.0001,hidden_layer_sizes=(100,100,100),
    learning_rate_init=0.1,verbose=10,solver='lbfgs', random_state=21, max_iter
    =50)
model201.fit(X201_train, Y201_train);
accuracy_OS_MLP_201=model201.score(X201_test,Y201_test)
print('Accuracy_MLP offset4:',accuracy_OS_MLP_201)
prediction_MLP_model201=model201.predict(X1_test)
MCC_MLP_model201 = matthews_corrcoef(Y1_test, prediction_MLP_model201)
print ('MCC_MLP_model201', MCC_MLP_model201)
print('offset_50:',MCC_clf201, MCC_rfc201, MCC_MLP_SGD201, MCC_MLP_model201)
OS_MCC=[MCC_clf201, MCC_rfc201, MCC_MLP_SGD201, MCC_MLP_model201]
%%% Fault Probability %%%
Sample_size=235
Fault_Rate1=10*235/100/235
print('Fault Rate1:', Fault_Rate1)
Fault_Rate2=20*235/100/235
print('Fault Rate2:', Fault_Rate2)
Fault_Rate3=30*235/100/235
print('Fault Rate3:', Fault_Rate3)
Fault_Rate4=40*235/100/235
print('Fault Rate4:', Fault_Rate4)
Fault_Rate5=50*235/100/235
print('Fault Rate5:', Fault_Rate5)
FAULT_PROBABILITY=[Fault_Rate1,Fault_Rate2,Fault_Rate3,Fault_Rate4,Fault_Rate5]
Accuracy_gain=[accuracy_gain_1*100, accuracy_gain_2*100, accuracy_gain_3*100,
    accuracy_gain_4*100, accuracy_gain_5*100]
Accuracy_OFB=[accuracy_OFB_11*100, accuracy_OFB_12*100,accuracy_OFB_13*100,
    accuracy_OFB_14*100,accuracy_OFB_15*100]
Accuracy_stuckat=[accuracy_SAT_16*100,accuracy_SAT_17*100,accuracy_SAT_18*100,
    accuracy_SAT_19*100,accuracy_SAT_20*100]
Accuracy_offset=[accuracy_OS_161*100,accuracy_OS_171*100,accuracy_OS_181*100,
    accuracy_OS_191*100,accuracy_OS_201*100]
```

Detection Accuracy of ERF Classifier

```
Accuracy_gain_RFC=[accuracy_gain_RFC_1*100, accuracy_gain_RFC_2*100,
    accuracy_gain_RFC_3*100, accuracy_gain_RFC_4*100, accuracy_gain_RFC_5*100]
Accuracy_OFB_RFC=[accuracy_OFB_RFC_11*100, accuracy_OFB_RFC_12*100,
    accuracy_OFB_RFC_13*100,accuracy_OFB_RFC_14*100,accuracy_OFB_RFC_15*100]
Accuracy_stuckat_RFC=[accuracy_SAT_RFC_16*100,accuracy_SAT_RFC_17*100,
    accuracy_SAT_RFC_18*100,accuracy_SAT_RFC_19*100,accuracy_SAT_RFC_20*100]
Accuracy_offset_RFC=[accuracy_OS_RFC_161*100,accuracy_OS_RFC_171*100,
    accuracy_OS_RFC_181*100,accuracy_OS_RFC_191*100,accuracy_OS_RFC_201*100]
```

Detection Accuracy SGD Classifier

```
Accuracy_gain_SGD=[accuracy_gain_SGD_1*100, accuracy_gain_SGD_2*100,
    accuracy_gain_SGD_3*100, accuracy_gain_SGD_4*100, accuracy_gain_SGD_5*100]
Accuracy_OFB_SGD=[accuracy_OFB_SGD_11*100, accuracy_OFB_SGD_12*100,
    accuracy_OFB_SGD_13*100,accuracy_OFB_SGD_14*100,accuracy_OFB_SGD_15*100]
Accuracy_stuckat_SGD=[accuracy_SAT_SGD_16*100,accuracy_SAT_SGD_17*100,
    accuracy_SAT_SGD_18*100,accuracy_SAT_SGD_19*100,accuracy_SAT_SGD_20*100]
Accuracy_offset_SGD=[accuracy_OS_SGD_161*100,accuracy_OS_SGD_171*100,
    accuracy_OS_SGD_181*100,accuracy_OS_SGD_191*100,accuracy_OS_SGD_201*100]
```

Detection Accuracy MLP Classifier

```
Accuracy_gain_MLP=[accuracy_gain_MLP_1*100, accuracy_gain_MLP_2*100,
    accuracy_gain_MLP_3*100, accuracy_gain_MLP_4*100, accuracy_gain_MLP_5*100]
Accuracy_OFB_MLP=[accuracy_OFB_MLP_11*100, accuracy_OFB_MLP_12*100,
    accuracy_OFB_MLP_13*100,accuracy_OFB_MLP_14*100,accuracy_OFB_MLP_15*100]
Accuracy_stuckat_MLP=[accuracy_SAT_MLP_16*100,accuracy_SAT_MLP_17*100,
    accuracy_SAT_MLP_18*100,accuracy_SAT_MLP_19*100,accuracy_SAT_MLP_20*100]
Accuracy_offset_MLP=[accuracy_OS_MLP_161*100,accuracy_OS_MLP_171*100,
    accuracy_OS_MLP_181*100,accuracy_OS_MLP_191*100,accuracy_OS_MLP_201*100]
%%% Printing all accuracies
print('All the Accuracy of SVC:',Accuracy_offset, Accuracy_gain, Accuracy_stuckat
    , Accuracy_OFB)
print('All the Accuracy of RFC:',Accuracy_offset_RFC, Accuracy_gain_RFC,
    Accuracy_stuckat_RFC, Accuracy_OFB_RFC)
print('All the Accuracy of SGD:',Accuracy_offset_SGD, Accuracy_gain_SGD,
    Accuracy_stuckat_SGD, Accuracy_OFB_SGD)
print('All the Accuracy of MLP:',Accuracy_offset_MLP, Accuracy_gain_MLP,
    Accuracy_stuckat_MLP, Accuracy_OFB_MLP)
```

Detection Accuracy GWO-SVM

```
fig=plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(78,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.plot(FAULT_PROBABILITY,Accuracy_offset,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat,'--^g', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_OFB,'--xm', linewidth=2.0)
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds')
    ,fontsize='medium');
plt.ylabel('GWO-SVM Detection Accuracy', fontsize=25); plt.xlabel('Fault
    Probability', fontsize=25);
plt.show()
%% Plotting RFC Detection Accuracy
fig = plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(78,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.plot(FAULT_PROBABILITY,Accuracy_offset_RFC,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain_RFC,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat_RFC,'--^g', linewidth=2.0)
```

```
plt.plot(FAULT_PROBABILITY,Accuracy_OFB_RFC,'--xm', linewidth=2.0)
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds'), fontsize='medium');
plt.ylabel(' ERF Detection Accuracy', fontsize=25); plt.xlabel('Fault Probability', fontsize=25);
plt.show()
%% Plotting SGD Detection Accuracy
fig = plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(65,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=23)
plt.rc('ytick', labelsize=23)
plt.plot(FAULT_PROBABILITY,Accuracy_offset_SGD,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain_SGD,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat_SGD,'--^g', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_OFB_SGD,'--xm', linewidth=2.0)
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds'), fontsize='medium');
plt.ylabel('SGD Detection Accuracy', fontsize=25); plt.xlabel('Fault Probability', fontsize=25);
plt.savefig('SGD_Detection_accuracy.eps',bbox_inches='tight',transparent='true')
plt.show()
%% Plotting MLP Detection Accuracy
fig = plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(78,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.plot(FAULT_PROBABILITY,Accuracy_offset_MLP,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain_MLP,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat_MLP,'--^g', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_OFB_MLP,'--xm', linewidth=2.0)
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds'));
plt.ylabel('MLP Detection Accuracy', fontsize=25); plt.xlabel('Fault Probability',fontsize=25);
plt.show()
```

Detection Accuracy DE-SVM

```
Cs = [1,10,100,1000,10000]
gammas = [.1,.01,.001,.0001,.00001]
tol=[0.001,.01,.1,.01,2]
param_grid = {'C': Cs, 'gamma' : gammas}
clf1 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf1.fit(X1_train, Y1_train)
prediction1=clf1.predict(X1_test)
accuracy_gain_1=clf1.score(X1_test, Y1_test)
print('Accuracy DESVM gain1:',accuracy_gain_1)
cm=confusion_matrix(Y1_test, prediction1)
print(cm)
MCC_1_SVM = matthews_corrcoef(Y1_test, prediction1)
print ('MCC_1_DESVM', MCC_1_SVM)
%%%%%%%% Apply DE_SVM for detection of GAIN FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
```

```
clf2 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf2.fit(X2_train, Y2_train)
accuracy_gain_2=clf2.score(X2_test, Y2_test)
print('Accuracy_ DESVM gain2:',accuracy_gain_2)
%%%%%%%% Apply DE_SVM for detection of GAIN FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf3 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf3.fit(X3_train, Y3_train)
accuracy_gain_3=clf3.score(X3_test, Y3_test)
print('Accuracy DESVM gain3:',accuracy_gain_3)
%%%%%%%% Apply DE_SVM for detection of GAIN FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf4 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf4.fit(X4_train, Y4_train)
accuracy_gain_4=clf4.score(X4_test, Y4_test)
print('Accuracy Desvm gain4:',accuracy_gain_4)
%%%%%%%% Apply DE_SVM for detection of GAIN FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf5 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf5.fit(X5_train, Y5_train)
accuracy_gain_5=clf5.score(X5_test, Y5_test)
print('Accuracy DESVM gain5:',accuracy_gain_5)
prediction_MLP_clf5=clf5.predict(X1_test)
MCC_1_MLP_5_clf_DESVM = matthews_corrcoef(Y1_test, prediction_MLP_clf5)
print ('MCC_DESVM', MCC_1_MLP_5_clf_DESVM)
%%%%%%%% Apply DE_SVM for detection of OUT OF BOUND FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf11 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf11.fit(X11_train, Y11_train)
accuracy_OFB_11=clf11.score(X11_test, Y11_test)
print('Accuracy DESVM ob1:',accuracy_OFB_11)
%%%%%%%% Apply DE_SVM for detection of OUT OF BOUND FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf12 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf12.fit(X12_train, Y12_train)
accuracy_OFB_12=clf12.score(X12_test, Y12_test)
print('Accuracy DESVM ob2:',accuracy_OFB_12)
%%%%%%%% Apply DE_SVM for detection of OUT OF BOUND FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf13 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf13.fit(X13_train, Y13_train)
accuracy_OFB_13=clf13.score(X13_test, Y13_test)
print('Accuracy DESVM ob3:',accuracy_OFB_13)
%%%%%%%% Apply DE_SVM for detection of OUT OF BOUND FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf14 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf14.fit(X14_train, Y14_train)
accuracy_OFB_14=clf14.score(X14_test, Y14_test)
print('Accuracy DESVM ob4:',accuracy_OFB_14)
%%%%%%%% Apply DE_SVM for detection of OUT OF BOUND FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf15 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf15.fit(X15_train, Y15_train)
accuracy_OFB_15=clf15.score(X15_test, Y15_test)
print('Accuracy DESVM ob5:',accuracy_OFB_15)
prediction_MLP_clf15=clf15.predict(X1_test)
MCC_MLP_15_clf_DESVM = matthews_corrcoef(Y1_test, prediction_MLP_clf15)
print ('MCC_DESVM', MCC_MLP_15_clf_DESVM)
```

```
%%%%%%%% Apply DE_SVM for detection of STUCKAT FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf16 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf16.fit(X16_train, Y16_train)
accuracy_SAT_16=clf16.score(X16_test, Y16_test)
print('Accuracy DESVM stuck1:',accuracy_SAT_16)
%%%%%%%% Apply DE_SVM for detection of STUCKAT FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf17 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf17.fit(X17_train, Y17_train)
accuracy_SAT_17=clf17.score(X17_test, Y17_test)
print('Accuracy DESVM stuck2:',accuracy_SAT_17)
%%%%%%%% Apply DE_SVM for detection of STUCKAT FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf18 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf18.fit(X18_train, Y18_train)
accuracy_SAT_18=clf18.score(X18_test, Y18_test)
print('Accuracy stuck3:',accuracy_SAT_18)
%%%%%%%% Apply DE_SVM for detection of STUCKAT FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf19 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf19.fit(X19_train, Y19_train)
accuracy_SAT_19=clf19.score(X19_test, Y19_test)
print('Accuracy stuck4:',accuracy_SAT_19)
%%%%%%%% Apply DE_SVM for detection of STUCKAT FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf20 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf20.fit(X20_train, Y20_train)
accuracy_SAT_20=clf20.score(X20_test, Y20_test)
print('Accuracy stuck5:',accuracy_SAT_20)
prediction_clf20=clf20.predict(X1_test)
MCC_clf20_DESVM = matthews_corrcoef(Y1_test, prediction_clf20)
print ('MCC_DESVM', MCC_clf20_DESVM)
%%%%%%%% Apply DE_SVM for detection of OFFSET FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf161 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf161.fit(X161_train, Y161_train)
accuracy_OS_161=clf161.score(X161_test, Y161_test)
print('Accuracy offset1:',accuracy_OS_161)
%%%%%%%% Apply DE_SVM for detection of OFFSET FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf171 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf171.fit(X171_train, Y171_train)
accuracy_OS_171=clf171.score(X171_test, Y171_test)
print('Accuracy offset2:',accuracy_OS_171)
%%%%%%%% Apply DE_SVM for detection of OFFSET FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf181 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf181.fit(X181_train, Y181_train)
accuracy_OS_181=clf181.score(X181_test, Y181_test)
print('Accuracy offset3:',accuracy_OS_181)
%%%%%%%% Apply DE_SVM for detection of OFFSET FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas}
clf191 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf191.fit(X191_train, Y191_train)
accuracy_OS_191=clf191.score(X191_test, Y191_test)
print('Accuracy offset4:',accuracy_OS_191)
%%%%%%%% Apply DE_SVM for detection of OFFSET FAULT in dataset 5 %%%%%%%%
```

```
param_grid = {'C': Cs, 'gamma' : gammas}
clf201 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf201.fit(X201_train, Y201_train)
accuracy_OS_201=clf201.score(X201_test, Y201_test)
print('Accuracy offset5:',accuracy_OS_201)
prediction_MLP_clf201=clf201.predict(X1_test)
MCC_clf201_DESVM = matthews_corrcoef(Y1_test, prediction_MLP_clf201)
print ('MCC_DESVM ', MCC_clf201_DESVM)
```

Plotting Detection Accuracy of DE-SVM

```
%%%Fault Probability
Sample_size=235
Fault_Rate1=10*235/100/235
print('Fault Rate1:', Fault_Rate1)
Fault_Rate2=20*235/100/235
print('Fault Rate2:', Fault_Rate2)
Fault_Rate3=30*235/100/235
print('Fault Rate3:', Fault_Rate3)
Fault_Rate4=40*235/100/235
print('Fault Rate4:', Fault_Rate4)
Fault_Rate5=50*235/100/235
print('Fault Rate5:', Fault_Rate5)
FAULT_PROBABILITY=[Fault_Rate1,Fault_Rate2,Fault_Rate3,Fault_Rate4,Fault_Rate5]
Accuracy_gain=[accuracy_gain_1*100, accuracy_gain_2*100, accuracy_gain_3*100,
    accuracy_gain_4*100, accuracy_gain_5*100]
Accuracy_OFB=[accuracy_OFB_11*100, accuracy_OFB_12*100,accuracy_OFB_13*100,
    accuracy_OFB_14*100,accuracy_OFB_15*100]
Accuracy_stuckat=[accuracy_SAT_16*100,accuracy_SAT_17*100,accuracy_SAT_18*100,
    accuracy_SAT_19*100,accuracy_SAT_20*100]
Accuracy_offset=[accuracy_OS_161*100,accuracy_OS_171*100,accuracy_OS_181*100,
    accuracy_OS_191*100,accuracy_OS_201*100]
print('All the Accuracy of DESVM:',Accuracy_offset, Accuracy_gain,
    Accuracy_stuckat, Accuracy_OFB)
%%%Plotting DE-SVM Detection Accuracy
fig = plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(78,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.plot(FAULT_PROBABILITY,Accuracy_offset,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat,'--^g', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_OFB,'--xm', linewidth=2.0)
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds'))
    ;
plt.ylabel('DE-SVM Detection Accuracy', fontsize=25); plt.xlabel('Fault
    Probability', fontsize=25);
plt.show()
%%%%%%%% Apply GA_SVM for detection of GAIN FAULT in dataset 1 %%%%%%%%
Cs = [2, 3, 1.9,3, 2]
gammas = [1.9,1.7, 1.5, 3]
tol=[0.001,.01,.1,.01,2]
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
```

```
clf1 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf1.fit(X1_train, Y1_train)
prediction1=clf1.predict(X1_test)
accuracy_gain_1=clf1.score(X1_test, Y1_test)
print('Accuracy GASVM gain1:',accuracy_gain_1)
cm=confusion_matrix(Y1_test, prediction1)
print(cm)
MCC_1_SVM = matthews_corrcoef(Y1_test, prediction1)
print ('MCC_1_GASVM', MCC_1_SVM)
%%%%%%%% Apply GA_SVM for detection of GAIN FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf2 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf2.fit(X2_train, Y2_train)
accuracy_gain_2=clf2.score(X2_test, Y2_test)
print('Accuracy GASVM_gain2:',accuracy_gain_2)
%%%%%%%% Apply GA_SVM for detection of GAIN FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf3 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf3.fit(X3_train, Y3_train)
accuracy_gain_3=clf3.score(X3_test, Y3_test)
print('Accuracy GASVM gain3:',accuracy_gain_3)
%%%%%%%% Apply GA_SVM for detection of GAIN FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf4 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf4.fit(X4_train, Y4_train)
accuracy_gain_4=clf4.score(X4_test, Y4_test)
print('Accuracy GASVM gain4:',accuracy_gain_4)
%%%%%%%% Apply GA_SVM for detection of GAIN FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf5 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf5.fit(X5_train, Y5_train)
accuracy_gain_5=clf5.score(X5_test, Y5_test)
print('Accuracy GASVM gain5:',accuracy_gain_5)
prediction_MLP_clf5=clf5.predict(X1_test)
MCC_1_MLP_5_clf_GASVM = matthews_corrcoef(Y1_test, prediction_MLP_clf5)
print ('MCC_GASVM', MCC_1_MLP_5_clf_GASVM)
%%%%%%%% Apply GA_SVM for detection of OUT OF BOUND FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf11 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf11.fit(X11_train, Y11_train)
accuracy_OFB_11=clf11.score(X11_test, Y11_test)
print('Accuracy ob1:',accuracy_OFB_11)
%%%%%%%% Apply GA_SVM for detection of OUT OF BOUND FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf12 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf12.fit(X12_train, Y12_train)
accuracy_OFB_12=clf12.score(X12_test, Y12_test)
print('Accuracy ob2:',accuracy_OFB_12)
%%%%%%%% Apply GA_SVM for detection of OUT OF BOUND FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf13 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf13.fit(X13_train, Y13_train)
accuracy_OFB_13=clf13.score(X13_test, Y13_test)
print('Accuracy ob3:',accuracy_OFB_13)
%%%%%%%% Apply GA_SVM for detection of OUT OF BOUND FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf14 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf14.fit(X14_train, Y14_train)
```

```
accuracy_OFB_14=clf14.score(X14_test, Y14_test)
print('Accuracy ob4:',accuracy_OFB_14)
%%%%%%%% Apply GA_SVM for detection of OUT OF BOUND FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf15 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf15.fit(X15_train, Y15_train)
accuracy_OFB_15=clf15.score(X15_test, Y15_test)
print('Accuracy ob5:',accuracy_OFB_15)
prediction_MLP_clf15=clf15.predict(X1_test)
MCC_MLP_15_clf_GASVM = matthews_corrcoef(Y1_test, prediction_MLP_clf15)
print ('MCC_GASVM', MCC_MLP_15_clf_GASVM)
%%%%%%%% Apply GA_SVM for detection of STUCKAT FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf16 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf16.fit(X16_train, Y16_train)
accuracy_SAT_16=clf16.score(X16_test, Y16_test)
print('Accuracy GASVM stuck1:',accuracy_SAT_16)
%%%%%%%% Apply GA_SVM for detection of STUCKAT FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf17 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf17.fit(X17_train, Y17_train)
accuracy_SAT_17=clf17.score(X17_test, Y17_test)
print('Accuracy GASVM stuck2:',accuracy_SAT_17)
%%%%%%%% Apply GA_SVM for detection of STUCKAT FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf18 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf18.fit(X18_train, Y18_train)
accuracy_SAT_18=clf18.score(X18_test, Y18_test)
print('Accuracy GASVM stuck3:',accuracy_SAT_18)
%%%%%%%% Apply GA_SVM for detection of STUCKAT FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf19 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf19.fit(X19_train, Y19_train)
accuracy_SAT_19=clf19.score(X19_test, Y19_test)
print('Accuracy GASVM stuck4:',accuracy_SAT_19)
%%%%%%%% Apply GA_SVM for detection of STUCKAT FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf20 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf20.fit(X20_train, Y20_train)
accuracy_SAT_20=clf20.score(X20_test, Y20_test)
print('Accuracy GASVM stuck5:',accuracy_SAT_20)
prediction_clf20=clf20.predict(X1_test)
MCC_clf20_GASVM = matthews_corrcoef(Y1_test, prediction_clf20)
print ('MCC_GASVM', MCC_clf20_GASVM)
%%%%%%%% Apply GA_SVM for detection of OFFSET FAULT in dataset 1 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf161 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf161.fit(X161_train, Y161_train)
accuracy_OS_161=clf161.score(X161_test, Y161_test)
print('Accuracy GASVM offset1:',accuracy_OS_161)
%%%%%%%% Apply GA_SVM for detection of OFFSET FAULT in dataset 2 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf171 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf171.fit(X171_train, Y171_train)
accuracy_OS_171=clf171.score(X171_test, Y171_test)
print('Accuracy GASVM offset2:',accuracy_OS_171)
%%%%%%%% Apply GA_SVM for detection of OFFSET FAULT in dataset 3 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
```

```
clf181 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf181.fit(X181_train, Y181_train)
accuracy_OS_181=clf181.score(X181_test, Y181_test)
print('Accuracy GASVM offset3:',accuracy_OS_181)
%%%%%%%% Apply GA_SVM for detection of OFFSET FAULT in dataset 4 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf191 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf191.fit(X191_train, Y191_train)
accuracy_OS_191=clf191.score(X191_test, Y191_test)
print('Accuracy GASVM offset4:',accuracy_OS_191)
%%%%%%%% Apply GA_SVM for detection of OFFSET FAULT in dataset 5 %%%%%%%%
param_grid = {'C': Cs, 'gamma' : gammas, 'tol': tol}
clf201 = GridSearchCV(SVC(C=1), param_grid, cv=KFold(2))
clf201.fit(X201_train, Y201_train)
accuracy_OS_201=clf201.score(X201_test, Y201_test)
print('Accuracy GASVMM offset5:',accuracy_OS_201)
prediction_MLP_clf201=clf201.predict(X1_test)
MCC_clf201_GASVM = matthews_corrcoef(Y1_test, prediction_MLP_clf201)
print ('MCC_GASVM ', MCC_clf201_GASVM)
```

Plotting Detection Accuracy of GA-SVM

```
%%%Fault Probability
Sample_size=400
Fault_Rate1=10*400/100/400
print('Fault Rate1:', Fault_Rate1)
Fault_Rate2=20*400/100/400
print('Fault Rate2:', Fault_Rate2)
Fault_Rate3=30*400/100/400
print('Fault Rate3:', Fault_Rate3)
Fault_Rate4=40*400/100/400
print('Fault Rate4:', Fault_Rate4)
Fault_Rate5=50*400/100/400
print('Fault Rate5:', Fault_Rate5)
FAULT_PROBABILITY=[Fault_Rate1,Fault_Rate2,Fault_Rate3,Fault_Rate4,Fault_Rate5]
Accuracy_gain=[accuracy_gain_1*100, accuracy_gain_2*100, accuracy_gain_3*100,
    accuracy_gain_4*100, accuracy_gain_5*100]
Accuracy_OFB=[accuracy_OFB_11*100, accuracy_OFB_12*100,accuracy_OFB_13*100,
    accuracy_OFB_14*100,accuracy_OFB_15*100]
Accuracy_stuckat=[accuracy_SAT_16*100,accuracy_SAT_17*100,accuracy_SAT_18*100,
    accuracy_SAT_19*100,accuracy_SAT_20*100]
Accuracy_offset=[accuracy_OS_161*100,accuracy_OS_171*100,accuracy_OS_181*100,
    accuracy_OS_191*100,accuracy_OS_201*100]
print('All the Accuracy of GASVM:',Accuracy_offset, Accuracy_gain,
    Accuracy_stuckat, Accuracy_OFB)
%%%Plotting Detection Accuracy of  GA-SVM
fig = plt.figure(figsize=(8,5))
plt.grid();
plt.ylim(78,100.9)
plt.xlim(0.05,0.55)
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
#plt.title("Detection accuracy of SVM according to fault type.")
plt.plot(FAULT_PROBABILITY,Accuracy_offset,'--Db', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_gain,'--sr', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_stuckat,'--^g', linewidth=2.0)
plt.plot(FAULT_PROBABILITY,Accuracy_OFB,'--xm', linewidth=2.0)
```

```
plt.gca().legend(('Offset Fault','Gain Fault','Stuck-at Fault', 'Out of Bounds'))
    ;
plt.ylabel('GA-SVM Detection Accuracy', fontsize=25); plt.xlabel('Fault
    Probability', fontsize=25);
plt.savefig('GA-SVM_Detection_accuracy.eps',bbox_inches='tight',transparent='true
    ')
plt.show()
%%%Average accuracies of all classifiers in detection of Gain Fault
Accuracy_Gain_GWOSVM_SUM=(accuracy_gain_1+accuracy_gain_2+accuracy_gain_3+
    accuracy_gain_4+accuracy_gain_5)/5
Accuracy_Gain_RF_SUM=(accuracy_gain_RFC_1+accuracy_gain_RFC_2+accuracy_gain_RFC_3
    +accuracy_gain_RFC_4+accuracy_gain_RFC_5)/5
Accuracy_Gain_SGD_SUM=(accuracy_gain_SGD_1+accuracy_gain_SGD_2+
    accuracy_gain_SGD_3+accuracy_gain_SGD_4+accuracy_gain_SGD_5)/5
Accuracy_Gain_MLP_SUM=(accuracy_gain_MLP_1+accuracy_gain_MLP_2+
    accuracy_gain_MLP_3+accuracy_gain_MLP_4+accuracy_gain_MLP_5)/5
Accuracy_Gain_DESVM_SUM=(accuracy_gain_1+accuracy_gain_2+accuracy_gain_3+
    accuracy_gain_4+accuracy_gain_5)/5
Accuracy_Gain_GASVM_SUM=(accuracy_gain_1+accuracy_gain_2+accuracy_gain_3+
    accuracy_gain_4+accuracy_gain_5)/5
%%%Average accuracies of all classifiers in detection of Out of bound fault
Accuracy_OB_GWOSVM_SUM=(accuracy_OFB_11+accuracy_OFB_12+accuracy_OFB_13+
    accuracy_OFB_14+accuracy_OFB_15)/5
Accuracy_OB_RF_SUM=(accuracy_OFB_RFC_11+accuracy_OFB_RFC_12+accuracy_OFB_RFC_13+
    accuracy_OFB_RFC_14+accuracy_OFB_RFC_15)/5
Accuracy_OB_SGD_SUM=(accuracy_OFB_SGD_11+accuracy_OFB_SGD_12+accuracy_OFB_SGD_13+
    accuracy_OFB_SGD_14+accuracy_OFB_SGD_15)/5
Accuracy_OB_MLP_SUM=(accuracy_OFB_MLP_11+accuracy_OFB_MLP_12+accuracy_OFB_MLP_13+
    accuracy_OFB_MLP_14+accuracy_OFB_MLP_15)/5
Accuracy_OB_DESVM_SUM=(accuracy_OFB_11+accuracy_OFB_12+accuracy_OFB_13+
    accuracy_OFB_14+accuracy_OFB_15)/5
Accuracy_OB_GASVM_SUM=(accuracy_OFB_11+accuracy_OFB_12+accuracy_OFB_13+
    accuracy_OFB_14+accuracy_OFB_15)/5
%%%Average accuracies of all classifiers in detection of Stuck at fault
Accuracy_stuck_GWOSVM_SUM=(accuracy_SAT_16+accuracy_SAT_17+accuracy_SAT_18+
    accuracy_SAT_19+accuracy_SAT_20)/5
Accuracy_stuck_RF_SUM=(accuracy_SAT_RFC_16+accuracy_SAT_RFC_17+
    accuracy_SAT_RFC_18+accuracy_SAT_RFC_19+accuracy_SAT_RFC_20)/5
Accuracy_stuck_SGD_SUM=(accuracy_SAT_SGD_16+accuracy_SAT_SGD_17+
    accuracy_SAT_SGD_18+accuracy_SAT_SGD_19+accuracy_SAT_SGD_20)/5
Accuracy_stuck_MLP_SUM=(accuracy_SAT_MLP_16+accuracy_SAT_MLP_17+
    accuracy_SAT_MLP_18+accuracy_SAT_MLP_19+accuracy_SAT_MLP_20)/5
Accuracy_stuck_DESVM_SUM=(accuracy_SAT_16+accuracy_SAT_17+accuracy_SAT_18+
    accuracy_SAT_19+accuracy_SAT_20)/5
Accuracy_stuck_GASVM_SUM=(accuracy_SAT_16+accuracy_SAT_17+accuracy_SAT_18+
    accuracy_SAT_19+accuracy_SAT_20)/5
%%%Average accuracies of all classifiers in detection of Offset Fault
Accuracy_offset_GWOSVM_SUM=(accuracy_OS_161+accuracy_OS_171+accuracy_OS_181+
    accuracy_OS_191+accuracy_OS_201)/5
Accuracy_offset_RF_SUM=(accuracy_OS_RFC_161+accuracy_OS_RFC_171+
    accuracy_OS_RFC_181+accuracy_OS_RFC_191+accuracy_OS_RFC_201)/5
Accuracy_offset_SGD_SUM=(accuracy_OS_SGD_161+accuracy_OS_SGD_171+
    accuracy_OS_SGD_181+accuracy_OS_SGD_191+accuracy_OS_SGD_201)/5
Accuracy_offset_MLP_SUM=(accuracy_OS_MLP_161+accuracy_OS_MLP_171+
    accuracy_OS_MLP_181+accuracy_OS_MLP_191+accuracy_OS_MLP_201)/5
Accuracy_offset_DESVM_SUM=(accuracy_OS_161+accuracy_OS_171+accuracy_OS_181+
    accuracy_OS_191+accuracy_OS_201)/5
```

```
Accuracy_offset_GASVM_SUM=(accuracy_OS_161+accuracy_OS_171+accuracy_OS_181+
    accuracy_OS_191+accuracy_OS_201)/5
```

True Positive Rate

```
GWOSVM_FaultsSUM=(Accuracy_Gain_GWOSVM_SUM+Accuracy_OB_GWOSVM_SUM+
    Accuracy_stuck_GWOSVM_SUM+Accuracy_offset_GWOSVM_SUM)/4
RF_FaultsSUM=(Accuracy_Gain_RF_SUM+Accuracy_OB_RF_SUM+Accuracy_stuck_RF_SUM+
    Accuracy_offset_RF_SUM)/4
SGD_FaultsSUM=(Accuracy_Gain_SGD_SUM+Accuracy_OB_SGD_SUM+Accuracy_stuck_SGD_SUM+
    Accuracy_offset_SGD_SUM)/4
MLP_FaultsSUM=(Accuracy_Gain_MLP_SUM+Accuracy_OB_MLP_SUM+Accuracy_stuck_MLP_SUM+
    Accuracy_offset_MLP_SUM)/4
DESVM_FaultsSUM=(Accuracy_Gain_DESVM_SUM+Accuracy_OB_DESVM_SUM+
    Accuracy_stuck_DESVM_SUM+Accuracy_offset_DESVM_SUM)/4
GASVM_FaultsSUM=(Accuracy_Gain_GASVM_SUM+Accuracy_OB_GASVM_SUM+
    Accuracy_stuck_GASVM_SUM+Accuracy_offset_MLP_SUM)/4
print('GWOSVM_FaultsSUM',GWOSVM_FaultsSUM)
print('RF_FaultsSUM',RF_FaultsSUM)
print('SGD_FaultsSUM',SGD_FaultsSUM)
print('MLP_FaultsSUM',MLP_FaultsSUM)
print('DESVM_FaultsSUM',DESVM_FaultsSUM)
print('GASVM_FaultsSUM',GASVM_FaultsSUM)
Total_obs=400
GWOSVM = GWOSVM_FaultsSUM*100/Total_obs
RFC = RF_FaultsSUM*100/Total_obs
SGD = SGD_FaultsSUM*100/Total_obs
MLP = MLP_FaultsSUM*100/Total_obs
DESVM = DESVM_FaultsSUM*100/Total_obs
GASVM = GASVM_FaultsSUM*100/Total_obs
print('GWOSVM, RFC, SGD, MLP, DESVM, GASVM', GWOSVM, RFC, SGD, MLP, DESVM,GASVM)
TPR_GWOSVM = GWOSVM/Total_obs
TPR_RFC = RFC/Total_obs
TPR_SGD = SGD/Total_obs
TPR_MLP = MLP/Total_obs
TPR_DESVM = DESVM/Total_obs
TPR_GASVM = GASVM/Total_obs
print('TPR_GWOSVM,TPR_RFC, TPR_SGD, TPR_MLP, TPR_DESVM, TPR_GASVM', TPR_GWOSVM,
    TPR_RFC, TPR_SGD, TPR_MLP,  TPR_DESVM, TPR_GASVM)
fig = plt.figure(figsize=(10,5))
width=1/2
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.ylabel('True Positive Rate (TPR)', fontsize=25)
D = {u'GWOSVM':TPR_GWOSVM, u'RF': TPR_RFC, u'SGD':TPR_SGD, u'MLP':TPR_MLP, u'
    DESVM':TPR_DESVM,  u'GASVM':TPR_GASVM}
plt.bar(range(len(D)), D.values(), align='center')
plt.xticks(range(len(D)), D.keys())
plt.show()
```

.3 Implementation of Proposed Solution 2:

This code is developed by Hafiza Syeda Zainab Kazmi under the supervision of Dr. Nadeem Javaid. To execute the code for Blockchain in Chapter 4, copy the code from Appendix C and paste in REMIX IDE using the names given at the start of each function with .sol extension. If you need any help or have any query regarding the code execution, you can email me at zainab.kazmi13@gmail.com. You can find detailed guidelines in *readme.txt* file.

Patient Monitoring Smart Contract

```
%%% Health Monitoring Smart Contract %%%
pragma solidity ^0.4.0;
```

Heart Rate Monitor Modular Contract

```
%%% Sub Contract Heart Rate Monitor %%%
contract HeartRateMonitor {


    function analyze(uint bpm, uint min, uint max) public constant returns (uint)
    {
        uint x=5;
        if(bpm < min||bpm > max){
            if(bpm < min-20||bpm > max+20){
                x=2;
                return (x);
            }
            x=1;
            return (x);
        }
        else{
            x=0;
            return (x);
        }
    }
}
```

Glucose Monitor Modular Contract

```
%%% Sub Contract Glucose Monitor %%%
contract GlucoseMonitor {
    function analyze(uint glucoseLevel, uint low, uint high) public constant
    returns (uint){
        uint x=5;
        if(glucoseLevel < low||glucoseLevel > high){
            if(glucoseLevel < low-20||glucoseLevel > high+80){
                if(glucoseLevel > high+140){
                    x=3;
                    return (x);
                }
                x=2;
```

```
                return (x);
            }
            x=1;
            return (x);
        }
        else{
            x=0;
            return (x);
        }
    }
}
```

Blood Pressure Monitor Modular Contract

```
%%% Sub Contract Blood Pressure Monitor %%%
contract BloodPressureMonitor {
    function analyze(uint bp, uint lo, uint hi) public constant returns (uint){
        uint x=5;
        if(bp < lo||bp > hi){
            if(bp < lo-40||bp > hi+90){
                if(bp > hi+150){
                    x=3;
                    return (x);
                }
                x=2;
                return (x);
            }
            x=1;
            return (x);
        }
        else{
            x=0;
            return (x);
        }
    }
}
```

Temprature Monitor Modular Contract

```
%%% Sub Contract Temprature Monitor %%%
contract TempratureMonitor {
    function analyze(uint t, uint ll, uint hh) public constant returns (uint){
        uint x=5;
        if(t < ll||t > hh){
            if(t < ll-20||t > hh+80){
                if(t > hh+100){
                    x=3;
                    return (x);
                }
                x=2;
                return (x);
            }
            x=1;
            return (x);
        }
        else{
```

```
                x=0;
                return (x);
            }
        }
}
```

Blood Oxygen Monitor Modular Contract

```
%%% Sub Contract Blood Oxygen Monitor %%%
contract BloodOxygenMonitor {
    function analyze(uint o, uint lll, uint hhh) public constant returns (uint){
        uint x=5;
        if(o < lll||o > hhh){
            if(o < lll-10||o > hhh+50){
                if(o > hhh+100){
                    x=3;
                    return (x);
                }
                x=2;
                return (x);
            }
            x=1;
            return (x);
        }
        else{
            x=0;
            return (x);
        }
    }
}
```

EEG Monitor Modular Contract

```
%%% Sub Contract EEG Monitor %%%
contract EEGMonitor {
    function analyze(uint e, uint llll, uint hhhh) public constant returns (uint)
    {
        uint x=5;
        if(e< llll||e > hhhh){
            if(e < llll-100||e > hhhh+200){
                if(e > hhhh+300){
                    x=3;
                    return (x);
                }
                x=2;
                return (x);
            }
            x=1;
            return (x);
        }
        else{
            x=0;
            return (x);
        }
    }
}
```

Main Patient Monitoring Smart Contract

```
%%% Main Contract of Health Monitoring that calls all sub contracts%%%
contract HealthContractCaller{

    function heartRateMonitor(uint bpm, uint min, uint max)public constant
    returns (uint code){

        HeartRateMonitor hrm = new HeartRateMonitor();

        return hrm.analyze(bpm, min, max);
    }

    function glucoseMonitor(uint glucoseLevel, uint low, uint high)public
    constant returns (uint code){

        GlucoseMonitor gm = new GlucoseMonitor();

        return gm.analyze(glucoseLevel, low, high);
    }

     function bloodPressureMonitor(uint bp, uint lo, uint hi)public constant
    returns (uint code){

        BloodPressureMonitor bpp = new BloodPressureMonitor();

        return bpp.analyze(bp, lo, hi);
    }

    function tempratureMonitor(uint t, uint ll, uint hh)public constant returns (
    uint code){

        TempratureMonitor temp = new TempratureMonitor();

        return temp.analyze(t, ll, hh);
    }
    function bloodOxygenMonitorMonitor(uint o, uint lll, uint hhh)public constant
     returns (uint code){

        BloodOxygenMonitor oxy = new BloodOxygenMonitor();

        return oxy.analyze(o, lll, hhh);
    }
    function eEGMonitor(uint e, uint llll, uint hhhh)public constant returns (
    uint code){

        EEGMonitor ee = new EEGMonitor();

        return ee.analyze(e, llll, hhhh);
    }

}
```

Enrollments Smart Contract

```
%%% Enrollments Smart Contract %%%
contract Enrollments {
    address public Hospital;
```

```
    modifier onlyHospital() {
        require(msg.sender == Hospital);
        _;
    }

    function Enrollments() public {
        Hospital = msg.sender;
    }
%%%%Paitent Enrollment%%%%%
    uint    public NumberOfPatients;
    mapping (address => bool)   public Patient_Account_IsRegistered;
    uint    public Patient_Id;
    event Patient_Added(address _address,uint _Patient_ID,string _Patient_Name,
    uint8 _Patient_Age,string _Patient_Address);
    event Patient_Modified(address _address,string _Patient_Name, uint8
    _Patient_Age,string _Patient_Address);
    struct Patient {
        address Patient_Account;
        uint    Patient_ID;
        string  Patient_Name;
        uint8   Patient_Age;
        string  Patient_Address;
    }
    mapping (address => Patient) patients;
    function Add_Patient(address _address,string _Patient_Name, uint8
    _Patient_Age,string _Patient_Address) onlyHospital public {
        require(_address != 0);
        require(Patient_Account_IsRegistered[_address] != true);
        require(Doctor_Account_IsRegistered[_address] != true);
        Patient_Account_IsRegistered[_address] = true;
        var patient             = patients[_address];
        patient.Patient_Account = _address;
        Patient_Id++;
        patient.Patient_ID      = Patient_Id;
        patient.Patient_Name    = _Patient_Name;
        patient.Patient_Age     = _Patient_Age;
        patient.Patient_Address = _Patient_Address;
        NumberOfPatients++;
        Patient_Added(_address, Patient_Id,_Patient_Name,_Patient_Age,
    _Patient_Address);
    }
    function Modify_Patient(address _address,string _Patient_Name, uint8
    _Patient_Age,string _Patient_Address) onlyHospital public {
        require(Patient_Account_IsRegistered[_address] == true);
        patients[_address].Patient_Name     = _Patient_Name;
        patients[_address].Patient_Age      = _Patient_Age;
        patients[_address].Patient_Address  = _Patient_Address;
        Patient_Modified(_address,_Patient_Name,_Patient_Age,_Patient_Address);
    }
    function PatientDetails(address _address) view public returns (address, uint,
     string, uint8, string)
        require(Patient_Account_IsRegistered[_address]);
        require((msg.sender == Hospital)||(listpatientfordoctors[msg.sender].
    Patient_Account_IsAuthorized[_address]==true)|| (msg.sender == _address));
        return (patients[_address].Patient_Account,patients[_address].Patient_ID,
     patients[_address].Patient_Name, patients[_address].Patient_Age, patients[
    _address].Patient_Address);
    }
```

```
%%%%%Medical Assistant Enrollment%%%%%
 uint    public NumberOfDoctors;
 mapping (address => bool) public Doctor_Account_IsRegistered;
 uint    public Doctor_Id;
 event Doctor_Added(address _address,uint _Doctor_ID,string _Doctor_Name,
 uint8 _Doctor_Age,string _Doctor_Address);
 event Doctor_Modified(address _address,string _Doctor_Name, uint8 _Doctor_Age
 ,string _Doctor_Address);
 struct Doctor {
     address Doctor_Account;
     uint    Doctor_ID;
     string  Doctor_Name;
     uint8   Doctor_Age;
     string  Doctor_Address;
 }
 mapping (address => Doctor) doctors;
 function Add_Doc(address _address,string _Doctor_Name, uint8 _Doctor_Age,
 string _Doctor_Address) onlyHospital public {
     require(_address != 0);
     require(Doctor_Account_IsRegistered[_address] != true);
     require(Patient_Account_IsRegistered[_address] != true);
     Doctor_Account_IsRegistered[_address] = true;
     var doctor              = doctors[_address];
     doctor.Doctor_Account   = _address;
     Doctor_Id++;
     doctor.Doctor_ID        = Doctor_Id;
     doctor.Doctor_Name      = _Doctor_Name;
     doctor.Doctor_Age       = _Doctor_Age;
     doctor.Doctor_Address   = _Doctor_Address;
     NumberOfDoctors++;
     Doctor_Added(_address, Doctor_Id,_Doctor_Name,_Doctor_Age,_Doctor_Address
 );
 }
 function Modify_Doctor(address _address,string _Doctor_Name, uint8
 _Doctor_Age,string _Doctor_Address) onlyHospital public {
     require(Doctor_Account_IsRegistered[_address] == true);
     doctors[_address].Doctor_Name       = _Doctor_Name;
     doctors[_address].Doctor_Age        = _Doctor_Age;
     doctors[_address].Doctor_Address    = _Doctor_Address;
     Doctor_Modified(_address,_Doctor_Name,_Doctor_Age,_Doctor_Address);
 }
 function Doctordetails(address _address) view public returns (address, uint,
 string, uint8, string) {
     require( Doctor_Account_IsRegistered[_address]);
     require((msg.sender == Hospital)||(msg.sender == _address));
     return (doctors[_address].Doctor_Account,doctors[_address].Doctor_ID,
 doctors[_address].Doctor_Name, doctors[_address].Doctor_Age, doctors[_address
 ].Doctor_Address);
 }
%%%% Patient Authorization %%%%
 struct ListPatientForDoctor {
     mapping (address => bool)  Patient_Account_IsAuthorized;
 }
 mapping (address => ListPatientForDoctor) listpatientfordoctors;
 function AuthorizePatient (address _Doctor_address,address _Patient_address)
 onlyHospital{
     require(Patient_Account_IsRegistered[_Patient_address] == true);
     require(Doctor_Account_IsRegistered[_Doctor_address] == true);
```

```
        var listpatientfordoctor            = listpatientfordoctors[
    _Doctor_address];
        listpatientfordoctor.Patient_Account_IsAuthorized[_Patient_address] =
    true;
    }
    function DeauthorizePatient (address _Doctor_address,address _Patient_address
    ) onlyHospital{
        require(Patient_Account_IsRegistered[_Patient_address] == true);
        require(Doctor_Account_IsRegistered[_Doctor_address] == true);
        var listpatientfordoctor            = listpatientfordoctors[
    _Doctor_address];
        listpatientfordoctor.Patient_Account_IsAuthorized[_Patient_address] =
    false;
    }
    function AuhtorizedPatientDetails (address _Doctor_address,address
    _Patient_address) onlyHospital view public returns(bool) {
        require(Patient_Account_IsRegistered[_Patient_address] == true);
        require(Doctor_Account_IsRegistered[_Doctor_address] == true);
        return (listpatientfordoctors[_Doctor_address].
    Patient_Account_IsAuthorized[_Patient_address]);
    }
     modifier onlyPatient() {
        require(Patient_Account_IsRegistered[msg.sender] == true);
        _;
    }
}
```

Enterprise Smart Contract

```
%%% Enterprise Smart Contract %%%
contract Enterprise {
    address[] public IoTDevices;
        address originalcustodian;
    modifier onlycustodian{
        require(msg.sender == originalcustodian);
        _;
        }
    event addNewIoT(string _DeviceName, address currentcustodian, address
    newDevicePA);
        function Enterprise(){
                originalcustodian = msg.sender;
        }
    function createIotContract (string _DeviceName, string _Devicedescription,
    string _serialNumber) onlycustodian returns (address){
        address newDevice = new IoTDevice(originalcustodian,  _DeviceName,
    _Devicedescription,  _serialNumber);
        IoTDevices.push(newDevice);
        addNewIoT(_DeviceName, msg.sender,newDevice);
        return newDevice;
    }
}
```

IoTDevice Authorization Smart Contract

```
%%% IoTDevice Smart Contract %%%
contract IoTDevice{
```

```
        address public custodian;
        string public DeviceName;
        string[] public Devicedescription;
        string public serialNumber;
        event addNewcustodian(string _msg, address newcustodian);
        function IoTDevice(address _custodian, string _DeviceName, string
    _Devicedescription, string _serialNumber){
        custodian = _custodian;
        DeviceName = _DeviceName;
        Devicedescription.push(_Devicedescription);
    serialNumber = _serialNumber;
        addNewcustodian ('Device made!', custodian);
        }
        modifier ifcustodian(){//prerequisite
                require(msg.sender == custodian);
                _;
        }
        function TransferPossesion(address newcustodian) ifcustodian {
                custodian = newcustodian ;
        }
        function DeviceNameUpdate(string newDeviceName) ifcustodian{
                DeviceName = newDeviceName;
        }
        function DevicedescriptionUpdate(string newDevicedescription) ifcustodian
    {
                Devicedescription.push(newDevicedescription);
        }
}
```

EHR IPFS Storage Smart Contract

```
%%%  IPFS Storage EHR Smart Contract %%%
contract IPFSStorageEHR
{
  function IPFSStorageEHR() public{
      start_time = now;
  }
  uint start_time = now;
    struct Patient
    {
        bytes32 email_p;
        uint adhar_id;
    }
   event Execution_Time(string Funtion_Name, uint Execution_Time);
    struct Doctor
    {
        bytes32 email_d;
        uint adhar_id_d;
    }
    struct IpfsHash
    {
        bytes32 first;
        bytes32 second;
        bytes32 third;
    }
   struct EhrDocument
   {
```

```
        bytes32 uploadedBy;
        bytes32 belongsTo;
       bytes32 date;
        IpfsHash encryptedHash;

    }
     mapping(address=>Patient) public PatientStruct ;
     mapping(address=>Doctor) public DoctorStruct;
     mapping(bytes32=>address) public PatientAddressMap;
    mapping(address=>EhrDocument[]) public PatientDocs;
     mapping(address=>mapping(address=>uint)) patientgrantaccess;
     mapping(bytes32=>bytes32) public usernameEmail;
     mapping(bytes32=>address) public DoctorAddressMap;
     function setPatient(bytes32 email_id,uint adhar_id,bytes32 username) public
     returns(bool success)
     {
         PatientStruct[msg.sender].email_p=email_id;
         PatientStruct[msg.sender].adhar_id=adhar_id;
         PatientAddressMap[keccak256(username)]=msg.sender;
         usernameEmail[keccak256(username)]=email_id;
         emit Execution_Time("setPatient",(now - start_time));
         return true;
     }
     function getPatient(address patient_add) public constant returns(bytes32
     email_p,uint adhar_id)
     {
         uint end_time = now;
         emit Execution_Time("setDoctor",(end_time - start_time));
         return (PatientStruct[patient_add].email_p,PatientStruct[patient_add].
     adhar_id);
     }
     function setDoctor(bytes32 email_id,uint adhar_id,bytes32 username) public
     returns(bool success)
     {
         DoctorStruct[msg.sender].email_d=email_id;
         DoctorStruct[msg.sender].adhar_id_d=adhar_id;
         DoctorAddressMap[keccak256(username)]=msg.sender;
        uint end_time = now;
        emit Execution_Time("setDoctor",(end_time - start_time));
         return true;
     }
     function getDoctor(address doctor_add) public constant returns(bytes32
     email_d,uint adhar_id_d)
     {
         emit Execution_Time("getDoctor",(now - start_time));
         return(DoctorStruct[doctor_add].email_d,DoctorStruct[doctor_add].
     adhar_id_d);
     }
     function getPatientAddress(bytes32 username) public constant returns(address)
     {
         emit Execution_Time("getPatientAddress",(now - start_time));
         return PatientAddressMap[keccak256(username)];
     }
     function getDoctorAddress(bytes32 username) public constant returns(address)
     {
         emit Execution_Time("getDoctorAddress",(now - start_time));
         return DoctorAddressMap[keccak256(username)];
     }
```

```
    function getPatientEmail(bytes32 username) public constant returns(bytes32)
    {
        emit Execution_Time("getPatientEmail",(now - start_time));
        return usernameEmail[keccak256(username)];
    }

    function grantAccess(address doc_address) public  returns(bool)
    {

        patientgrantaccess[msg.sender][doc_address]=1;
        emit Execution_Time("grantAccess",(now - start_time));
        return true;
    }
    function checkstatus(address doc_address)view public returns(bool)
    {
        emit Execution_Time("checkstatus",(now - start_time));
        if(patientgrantaccess[msg.sender][doc_address]!=1)
        return true;
        else
        return false;
    }
    function checkstatusdoc(address patient_address) view public returns(bool)
    {
        emit Execution_Time("checkstatusdoc",(now - start_time));
        if(patientgrantaccess[patient_address][msg.sender]==1)
        return true;
        else
        return false;
    }
    function storeIpfs(bytes32 doc_username,bytes32 patient_username,bytes32 date
    ,bytes32 first,bytes32 second,bytes32 third) public returns(bool success)
    {
       EhrDocument storage doc;
        doc.uploadedBy=doc_username;
        doc.belongsTo=patient_username;
        doc.encryptedHash.first=first;
        doc.encryptedHash.second=second;
        doc.encryptedHash.third=third;
        doc.date=date;
        PatientDocs[msg.sender].push(doc);

      emit Execution_Time("storeIpfs",(now - start_time));
        return true;
    }

  function getEHRDetails(address patient_address) public constant returns(bytes32
    [],bytes32[],bytes32[],bytes32[],bytes32[] )
  {
      bytes32[] memory uploadedBy = new bytes32[](PatientDocs[patient_address].
    length);
      bytes32[] memory date = new bytes32[](PatientDocs[patient_address].length);
      bytes32[] memory part1 = new bytes32[](PatientDocs[patient_address].length)
    ;
      bytes32[] memory part2 = new bytes32[](PatientDocs[patient_address].length)
    ;
      bytes32[] memory part3 = new bytes32[](PatientDocs[patient_address].length)
      for (uint i=0; i < PatientDocs[patient_address].length ; i++)
      {
```

```
            EhrDocument storage doc = PatientDocs[patient_address][i];
            uploadedBy[i] = doc.uploadedBy;
            date[i] = doc.date;
            part1[i] = doc.encryptedHash.first;
            part2[i] = doc.encryptedHash.second;
            part3[i] = doc.encryptedHash.third;
        }
        emit Execution_Time("getEHRDetails",(now - start_time));
        return (uploadedBy, date, part1, part2, part3);
    }
}
```

Patient Review and Rating Smart Contract

```
%%% Patiient Review System Smart Contract %%%
contract Review{
    struct Data{
        string data_contents;
        int data_rating;
    }
    struct Writer_Reviews{
        mapping(bytes32 => Data) datas;
    }
address writer;
    uint public reviewCounter;
    mapping (address => Writer_Reviews) reviews;
    function IsReviewExist(bytes32 metadata) public view returns (int){
        address add = msg.sender;
        if(reviews[add].datas[metadata].data_rating==0){
            return 0;
        }
        else
        {
            return 1;
        }
    }
    function GiveReviews(bytes32 metadata, string memory data, string memory
    contents, int rating) public returns (int){
        writer = msg.sender;
        reviews[writer].datas[metadata].data_contents = contents;
        reviews[writer].datas[metadata].data_rating = rating;
        reviewCounter++;
        return 1;
    }
    function searchReview(bytes32 metadata) public view returns(string memory){
        return reviews[msg.sender].datas[metadata].data_contents;
    }

    function searchRatings(bytes32 metadata) public view returns (int) {
        return reviews[msg.sender].datas[metadata].data_rating;
    }

}
```

Encryption and Decryption

```
%%%%% Encryption  and Decryption Time Calculation%%%%%%%
%%%This is the python code used for the calculation of encryption and decryption
    time of algorithms offchain.
%%% Import packages %%%
import hashlib
import hmac
import Crypto
import Crypto.Cipher.AES
import Crypto.Util.Padding
import secrets
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES
from binascii import hexlify
from binascii import unhexlify
from pyDes import *
import pickle
import time
start = time. time()
def generate_master_key(algorithm_choice):
    %5Creation of master key using PBKDF#2 hashed with either SHA256 or SHA512.
    Salt is a randomly generated 16 characters in hex format.

    %Args:
       % algorithm_choice (integer):
           % An integer who's value determines which algorithm is
            %going to be used.

 %   Return:
      %  key (byte):
          %  The generated master key to be used for encryption and
            %hashing derivation.
    %%
    salt = str.encode(secrets.token_hex(8))
    if algorithm_choice == 1:
        key = hashlib.pbkdf2_hmac('sha256', b'>>$$MasterPassword9000$$<<', salt,
    100000)
    else:
        key = hashlib.pbkdf2_hmac('sha512', b'>>$$MasterPassword9000$$<<', salt,
    100000)
    return key


def generate_encryption_key(key_length=16):
    %Derivation of encryption key using PBKDF#2.
    %Hashed and salted.

    %Args:
        %key_length (integer):
           % Length of the key needed to be generated.
            %accepts multiples of 16, expecting values
            %of either 16 or 32.

    %Return:
        %key (byte):
            %The encryption key for each algorithm.

    salt = str.encode(secrets.token_hex(8))
```

```
    key = hashlib.pbkdf2_hmac('sha256', master_key, salt, 1, key_length)
    return key


def generate_hmac(key, data=b'123'):
    %Generate of the HMAC.

    %Args:
        %data (byte):
           % The cipher text to be hashed. Default data
            %to prevent errors.
        %key (byte):
            %The derived key from the master encryption key.

   % Return:
       % HMAC (byte):
           % The HMAC of the cipher text.

    return hmac.new(key, data, hashlib.sha256).hexdigest()


def generate_hmac_key(key_length=16):

    salt = str.encode(secrets.token_hex(8))
    key = hashlib.pbkdf2_hmac('sha256', master_key, salt, 1, key_length)
    return key


def hash_select():
    %Allows the user to be able to select between SHA126
  %  and SHA258 hashing algorithms.

  %  Args: None.

  %  Return:
       % key (byte):
         %   The master encryption key.
    %
    print('Would you like to use sha256 or sha512?')
    print('1. sha256')
    print('2. sha512')
    while True:
        try:
            hash_choice = int(input())
            if hash_choice == 1:
                break
            if hash_choice == 2:
                break
            print('Enter 1 or 2.')
        except ValueError:
            print("Please enter 1 or 2")
    key = ""
    if hash_choice == 1:
        key = generate_master_key(1)
    if hash_choice == 2:
        key = generate_master_key(2)
    return key
def generate_iv(block_size=56):
```

```
    %Generated random bytes of various block size to
  %  be used as an IV.
    %Args:
        %block_size (integer):
            %The size of the desired block.
    %Return:
        %random_bytes (byte):
            %"block_size" amount of randomly generated bytes
            %to be used as an injection vector.
    return get_random_bytes(block_size)
def encrypt_aes256(plaintext):
    %Implementation of AES256. Key size of 256 bits with
    %a block size of 126. PKCS7 padding. Encrypts using AES256
    %to a file. Additionally, an HMAC is generated to verify
    %data integrity.
    %Args:
        %plaintext (byte)
            %The plain text to be encrypted.
   % Return: None.
    % Initial set up of encryption cipher.
    algorithm = "aes256"
    key_size = 32  %256 bit key.
    block_size = 16
    encryption_key = generate_encryption_key(key_size)
    iv = generate_iv(block_size)
    plaintext = Crypto.Util.Padding.pad(plaintext, block_size, style='pkcs7')
    cipher = AES.new(encryption_key, AES.MODE_CBC, iv)
    % Encryption of data and generation of HMAC.
    ciphertext = cipher.encrypt(plaintext)
    local_hmac = generate_hmac(hmac_derived_key, ciphertext+iv)
    % Write encrypted data to file.
    try:
        f = open("encrypted.txt", "wb")
        f.write(hexlify(ciphertext))
        f.close()
    except FileNotFoundError:
        print("Can not find file!")
    % User feedback.
    print("NOW ENCRYPTING:" + algorithm)
    print("\n HMAC:\n" + local_hmac)
    print("\n Encrypted:")
    print(ciphertext)
    del ciphertext
    % Generate and serialize cipher metadata.
    local_keys = dict(int_list=[],
                      my_keys=encryption_key,
                      my_hmac=hmac_derived_key,
                      my_iv=iv,
                      my_block_size=block_size,
                      my_algorithm=algorithm,
                      my_key_size=key_size)

    with open('keys.pkl', 'wb') as f:
        pickle.dump(local_keys, f)
   % Generate HMAC file.
    try:
        f = open("hmac.txt", "w")
        f.write(local_hmac)
```

```
        f.close()
    except FileNotFoundError:
        print("Can not find file!")
%%%%%%%%%%%%%%  Encryption using 3DES andd AES256%%%%%%%%%%%%%%
def encrypt_3des(plaintext):
    %Implementation of 3DES. Key size of 126 bits with
    %a block size of 56 bits. PKCS7 padding. Encrypts using 3DES
    %to a file. Additionally, an HMAC is generated to verify
    %data integrity.
    %Args:
        %plaintext (byte):
            %The plain text to be encrypted.
    %Return: None.
    %Initial set up
    algorithm = "3des"
    key_size = 16
    block_size = 16
    iv = generate_iv(8)
    encryption_key = generate_encryption_key(block_size)
    print(plaintext.decode())
    plaintext = Crypto.Util.Padding.pad(plaintext, block_size, style='pkcs7')
    cipher = triple_des(encryption_key, CBC, iv, pad=None)
    % Encryption of data and generation of HMAC.
    ciphertext = cipher.encrypt(plaintext)
    local_hmac = generate_hmac(hmac_derived_key, ciphertext+iv)
    % Write encrypted data to file.
    f = open("encrypted.txt", "wb")
    f.write(hexlify(ciphertext))
    f.close()
    %User feedback.
    print("NOW ENCRYPTING WITH " + algorithm.upper() + ":")
    print("\n HMAC:\n" + local_hmac)
    print("\n Encrypted:")
    print(ciphertext)
    del ciphertext
    %Generate and serialize cipher metadata.
    local_keys = dict(int_list=[],
                      my_keys=encryption_key,
                      my_hmac=hmac_derived_key,
                      my_iv=iv,
                      my_block_size=block_size,
                      my_algorithm=algorithm,
                      my_key_size=key_size)

    with open('keys.pkl', 'wb') as f:
        pickle.dump(local_keys, f)
    f.close()
    %Generate HMAC file.
    try:
        f = open("hmac.txt", "w")
        f.write(local_hmac)
        f.close()
    except FileNotFoundError:
        print("Can not find file!")
%%%%%%%%%%%%%%  Decryption using 3DES andd AES256%%%%%%%%%%%%%%
def decrypt():
    %Decrypts from a text file cipher text that has been generated
    %using algorithms 3DES, AES128, or AES256. Reads metadata from
```

```
    %keys.pkl. Will detect algorithm used and send plaintext to
    %"plaintext.txt".
    %Args: None
    %Return: None
    %Initialize variables
    algorithm = "Unknown Algorithm"
    local_recovered_hmac_key = ""
    encryption_key = ""
    iv = ""
    block_size = ""
   % Unpack the data and ensure format is correct.
    try:
        with open('keys.pkl', 'rb') as f:
            enc_meta = pickle.load(f)
        encryption_key = enc_meta['my_keys']
        local_recovered_hmac_key = enc_meta['my_hmac']
        iv = enc_meta['my_iv']
        block_size = enc_meta['my_block_size']
        algorithm = enc_meta['my_algorithm']
    except (FileNotFoundError, RuntimeError):
        print("File format is incorrect. Encrypt the data using this program.")

    % Ensure it is a registered algorithm.
    if  algorithm != "aes256" and algorithm != "3des":
        print("Error trying to decrypt " + algorithm)
        sys.exit(0)

    %Opening file and reading ciphertext.
    print("NOW DECRYPTING WITH " + algorithm.upper() + ":")
    ciphertext = "Failed to load."
    try:
        f = open("encrypted.txt", "br")
        ciphertext = f.read()
        f.close()
    except FileNotFoundError:
        print("Can not find file!")

    % Generating HMAC
    local_hmac = generate_hmac(local_recovered_hmac_key, unhexlify(ciphertext)+iv
    )
    print("\n Generated HMAC:")
    print(local_hmac)

    %Reading HMAC generated at encryption time.
    test_hmac = "Failed to load."
    try:
        f = open("hmac.txt", "r")
        test_hmac = f.read()
        f.close()

    except FileNotFoundError:
        print("Can not find file!")

    print("\n Registered HMAC:")
    print(test_hmac)

    %Ensure match
    if test_hmac != local_hmac:
```

```
            print("\n CORRUPTED DATA: Alterations have been made!")
            sys.exit(0)
        else:
            print("\n MATCH")

        %Choose decryption algorithm.
        if  algorithm == "aes256":
            decipher = AES.new(encryption_key, AES.MODE_CBC, iv)
        else:
            decipher = triple_des(encryption_key, CBC, iv, pad=None)

        %Decrypt and decode.
        plaintext = decipher.decrypt(unhexlify(ciphertext))
        plaintext = Crypto.Util.Padding.unpad(plaintext, block_size, style='pkcs7')
        print("\n Decrypted:")
        try:
            f = open("plaintext.txt", "w")
            f.write(plaintext.decode())
            f.close()
        except FileNotFoundError:
            print("Can not find file!")
        print(plaintext.decode())


def user_choice():
    %Get the user's decision on if they want to encrypt
    %a file or decrypt one.

    %Args: None

    %Return:
        %users_choice (integer):
            %Numeric representation of the users choice.

    print("\n\n Would you like to encrypt or decrypt?")
    print("1. Encrypt")
    print("2. Decrypt")
    users_choice = get_int(1)
    if users_choice == 1:
        return 1
    if users_choice == 2:
        return 2


def get_int(self=1):
    %Safely retrieve a numeric representation of
    %a users choice to logic processing.

%    Args:
    %    self (integer):
        %    Control flow for the number of decisions needed.

    %Return:
        %users_choice (integer):
            %Numeric representation of the users choice.

    while True:
        if self == 1:
```

```
            try:
                users_choice = int(input())
                if users_choice == 1 or users_choice == 2:
                    break
                print('Enter 1 or 2')
            except ValueError:
                print('Enter 1 or 2')
        if self == 2:
            try:
                users_choice = int(input())
                if users_choice == 1\
                        or users_choice == 2 or users_choice == 3:
                    break
                print('Enter 1, 2 or 3')
            except ValueError:
                print('Enter 1, 2 or 3')
    return users_choice


if __name__ == '__main__':
    %Start
    choice = user_choice()

    % Encryption
    if choice == 1:
        try:
            unencrypted_text = (open('plaintext.txt', 'rb'))
            unencrypted_text = unencrypted_text.read()
            print()
            print("This is the plaintext to be encrypted:")
            print(unencrypted_text.decode())
            print()
        except FileNotFoundError:
            print("Ensure the text to be encrypted is in the local directory as
    \"plaintext.txt\"")
            sys.exit(0)
        master_key = hash_select()
        hmac_derived_key = generate_hmac_key()

        % User selection of encryption algorithm.
        print("Please select which algorithm you would like to use:")
        print("1. 3des")
        print("3. aes256")
        print()
        alg = get_int(2)
        if alg == 1:
            print(type(unencrypted_text))
            print(unencrypted_text)
            encrypt_3des(unencrypted_text)
        if alg == 3:
            encrypt_aes256(unencrypted_text)

    % Decryption
    if choice == 2:
        decrypt()
end = time. time()
print(end - start)
```

```
%%%%%%%%%%%%  Affine Cipher Encryption %%%%%%%%%%%%%%
import operator
from os import system
import time
start = time. time()

# characters to numbers tables
cnall = {’A’: 0, ’B’: 1, ’C’: 2, ’D’: 3, ’E’: 4, ’F’: 5, ’G’: 6, ’H’: 7, ’I’: 8,
         ’J’: 9, ’K’: 10, ’L’: 11, ’M’: 12, ’N’: 13, ’O’: 14, ’P’: 15, ’Q’: 16,
         ’R’: 17, ’S’: 18, ’T’: 19, ’U’: 20, ’V’: 21, ’W’: 22, ’X’: 23, ’Y’: 24,
         ’Z’: 25, ’a’: 26, ’b’: 27, ’c’: 28, ’d’: 29, ’e’: 30, ’f’: 31, ’g’: 32,
         ’h’: 33, ’i’: 34, ’j’: 35, ’k’: 36, ’l’: 37, ’m’: 38, ’n’: 39, ’o’: 40,
         ’p’: 41, ’q’: 42, ’r’: 43, ’s’: 44, ’t’: 45, ’u’: 46, ’v’: 47, ’w’: 48,
         ’x’: 49, ’y’: 50, ’z’: 51}

cnupper = {’A’: 0, ’B’: 1, ’C’: 2, ’D’: 3, ’E’: 4, ’F’: 5, ’G’: 6, ’H’: 7, ’I’:
    8,
           ’J’: 9, ’K’: 10, ’L’: 11, ’M’: 12, ’N’: 13, ’O’: 14, ’P’: 15, ’Q’: 16,
           ’R’: 17, ’S’: 18, ’T’: 19, ’U’: 20, ’V’: 21, ’W’: 22, ’X’: 23, ’Y’:
    24,
           ’Z’: 25}

# numbers to characters tables
ncall = {0: ’A’, 1: ’B’, 2: ’C’, 3: ’D’, 4: ’E’, 5: ’F’, 6: ’G’, 7: ’H’, 8: ’I’,
         9: ’J’, 10: ’K’, 11: ’L’, 12: ’M’, 13: ’N’, 14: ’O’, 15: ’P’, 16: ’Q’,
         17: ’R’, 18: ’S’, 19: ’T’, 20: ’U’, 21: ’V’, 22: ’W’, 23: ’X’, 24: ’Y’,
         25: ’Z’, 26: ’a’, 27: ’b’, 28: ’,c’, 29: ’d’, 30: ’e’, 31: ’f’, 32: ’g’,
         33: ’h’, 34: ’i’, 35: ’j’, 36: ’k’, 37: ’l’, 38: ’m’, 39: ’n’, 40: ’o’,
         41: ’p’, 42: ’q’, 43: ’r’, 44: ’s’, 45: ’t’, 46: ’u’, 47: ’v’, 48: ’w’,
         49: ’x’, 50: ’y’, 51: ’z’}

ncupper = {0: ’A’, 1: ’B’, 2: ’C’, 3: ’D’, 4: ’E’, 5: ’F’, 6: ’G’, 7: ’H’, 8: ’I
    ’,
           9: ’J’, 10: ’K’, 11: ’L’, 12: ’M’, 13: ’N’, 14: ’O’, 15: ’P’, 16: ’Q’,
           17: ’R’, 18: ’S’, 19: ’T’, 20: ’U’, 21: ’V’, 22: ’W’, 23: ’X’, 24: ’Y
    ’,
           25: ’Z’}


def crypted26(vigenere_text, shift_, multiplier_):
    crypt_Text = []

    for clearChar in map(operator.add, vigenere_text[::2], vigenere_text[1::2]):

        # loop characters in key
        a = clearChar

        i = 0
        j = 1
        while i < len(a):

            left = a[i]

            while j < len(a):
                right = a[j]
                h = 0
                char_num = str(cnall[left])
                if len(char_num) == 1:
```

```
                    char_num = str(h) + char_num
                # print(charNum)
                char_num1 = str(cnall[right])
                if len(char_num1) == 1:
                    char_num1 = str(h) + char_num1
                # print(charNum1)
                d = int(str(char_num) + str(char_num1))
                z = (int(multiplier_) * d)
                y = (z + int(shift_))
                crypt = (y % 2526)
                crypt_char = str(crypt)
                print(crypt_char)
                if len(crypt_char) == 3:
                    crypt_char = str(h) + crypt_char
                elif len(crypt_char) == 2:
                    crypt_char = str(h) + str(h) + crypt_char
                print(crypt_char)
                crypt_Text.append(crypt_char)

                break

            break
    return ''.join(crypt_Text)


def convert26(crypt_text):
    affine = []

    for char in map(operator.add, crypt_text[::2], crypt_text[1::2]):
        crypted1 = str(char)
        s = 0
        t = 1

        while s < len(crypted1):
            h = 0
            left = crypted1[s]
            if left == str(h):
                left = ''

            while t < len(crypted1):
                right = crypted1[t]

                combine = (str(left) + str(right))
                print(combine)
                lookup = ncall[int(combine)]
                affine.append(lookup)
                break
            break
    return ''.join(affine)


def crypted52(vigenere_, shift_, multiplier_):
    crypt_Text = []

    for clearChar in map(operator.add, vigenere_[::2], vigenere_[1::2]):
        # loop characters in key
        a = clearChar

```

```
        i = 0
        j = 1
        while i < len(a):

            left = a[i]

            while j < len(a):
                right = a[j]
                h = 0
                char_num = str(cnall[left])
                if len(char_num) == 1:
                    char_num = str(h) + char_num
                # print(charNum)
                char_num1 = str(cnall[right])
                if len(char_num1) == 1:
                    char_num1 = str(h) + char_num1
                # print(charNum1)
                d = int(str(char_num) + str(char_num1))
                z = (int(multiplier_) * d)
                y = (z + int(shift_))
                crypt = (y % 5152)
                crypt_char = str(crypt)
                print(crypt_char)
                if len(crypt_char) == 3:
                    crypt_char = str(h) + crypt_char
                elif len(crypt_char) == 2:
                    crypt_char = str(h) + str(h) + crypt_char
                print(crypt_char)
                crypt_Text.append(crypt_char)

                break

            break
    return ''.join(crypt_Text)


def convert52(crypt_text):
    affine = []

    for char in map(operator.add, crypt_text[::2], crypt_text[1::2]):
        crypted1 = str(char)
        s = 0
        t = 1

        while s < len(crypted1):
            h = 0
            left = crypted1[s]
            if left == str(h):
                left = ''

            while t < len(crypted1):
                right = crypted1[t]

                combine = (str(left) + str(right))
                print(combine)
                lookup = ncall[int(combine)]
                affine.append(lookup)
                break
```

```
                break
    return ''.join(affine)


while True:

    with open('F:\kazmi\Vigenere-and-block-affine-cipher-master/plaintext.txt', '
    r') as myfile:
        vigenere = myfile.read().replace('\n', '')
    print(vigenere)

    shift = input("Enter value of b\n\n")
    system('cls')

    multiplier = input("Enter value of m\n\n")
    system('cls')

    choice = input("Enter S or L\n\n")
    system('cls')

    if choice.lower() == 's':
        vigeneretext = vigenere.upper()
        cryptText = crypted26(vigeneretext, shift, multiplier)
        print(cryptText)
        AffineText = convert26(cryptText)
        print(AffineText)
        print('done')
        cryptDir = open('F:\kazmi\Vigenere-and-block-affine-cipher-master/
    plaintext.txt', 'w')
        cryptDir.write(cryptText)
        cryptDir.close()
        break

    elif choice.lower() == 'l':

        cryptText = crypted52(vigenere, shift, multiplier)
        print(cryptText)
        AffineText = convert52(cryptText)
        print(AffineText)
        print('done')
        cryptDir = open('F:\kazmi\Vigenere-and-block-affine-cipher-master/
    plaintext.txt', 'w')
        cryptDir.write(cryptText)
        cryptDir.close()
        break
    break
end = time. time()
print(end - start)
%%%%%%%%%%%%  Affine Cipher Decryption %%%%%%%%%%%%%%
import operator
from os import system
import time
start = time. time()

# characters to numbers tables
cnall = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8,
         'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16,
         'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24,
```

```python
          'Z': 25, 'a': 26, 'b': 27, 'c': 28, 'd': 29, 'e': 30, 'f': 31, 'g': 32,
          'h': 33, 'i': 34, 'j': 35, 'k': 36, 'l': 37, 'm': 38, 'n': 39, 'o': 40,
          'p': 41, 'q': 42, 'r': 43, 's': 44, 't': 45, 'u': 46, 'v': 47, 'w': 48,
          'x': 49, 'y': 50, 'z': 51}

cnupper = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I':
    8,
            'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16,
            'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y':
    24,
            'Z': 25}

# numbers to characters tables
ncall = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I',
          9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q',
          17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y',
          25: 'Z', 26: 'a', 27: 'b', 28: ',c', 29: 'd', 30: 'e', 31: 'f', 32: 'g',
          33: 'h', 34: 'i', 35: 'j', 36: 'k', 37: 'l', 38: 'm', 39: 'n', 40: 'o',
          41: 'p', 42: 'q', 43: 'r', 44: 's', 45: 't', 46: 'u', 47: 'v', 48: 'w',
          49: 'x', 50: 'y', 51: 'z'}

ncupper = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I
    ',
            9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q',
            17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y
    ',
            25: 'Z'}


def egcd(multiplier, modulus):
    if multiplier == 0:
        return modulus, 0, 1
    else:
        g, y, x = egcd(modulus % multiplier, multiplier)
        return g, x - (modulus // multiplier) * y, y


def modinv(multiplier, modulus):
    g, x, y = egcd(multiplier, modulus)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % modulus


def crypted26(affine_encrypttext, shift_1, minverse_1):
    crypt_Text = []

    for chars in map(operator.add, affine_encrypttext[::2], affine_encrypttext
    [1::2]):

        # loop characters in key
        a = chars

        i = 0
        j = 1
        while i < len(a):

```

```
            left = a[i]

            while j < len(a):
                right = a[j]
                h = 0
                char_num = str(cnall[left])
                if len(char_num) == 1:
                    char_num = str(h) + char_num
                # print(charNum)
                char_num1 = str(cnall[right])
                if len(char_num1) == 1:
                    char_num1 = str(h) + char_num1
                # print(charNum1)
                d = int(str(char_num) + str(char_num1))
                z = (d - int(shift_1))
                y = (int(minverse_1) * z)
                crypt = (y % 2526)
                crypt_char = str(crypt)
                print(crypt_char)
                if len(crypt_char) == 3:
                    crypt_char = str(h) + crypt_char
                elif len(crypt_char) == 2:
                    crypt_char = str(h) + str(h) + crypt_char
                print(crypt_char)
                crypt_Text.append(crypt_char)

                break

            break
    return ''.join(crypt_Text)


def convert26(crypt_text):
    affine = []

    for char in map(operator.add, crypt_text[::2], crypt_text[1::2]):
        crypted1 = str(char)
        s = 0
        t = 1

        while s < len(crypted1):
            h = 0
            left = crypted1[s]
            if left == str(h):
                left = ''

            while t < len(crypted1):
                right = crypted1[t]

                combine = (str(left) + str(right))
                print(combine)
                lookup = ncall[int(combine)]
                affine.append(lookup)
                break
            break
    return ''.join(affine)
```

```
def crypted52(affine_encrypt, shift_1, minverse_):
    crypt_Text = []

    for chars in map(operator.add, affine_encrypt[::2], affine_encrypt[1::2]):
        # loop characters in key
        a = chars
        i = 0
        j = 1
        while i < len(a):

            left = a[i]

            while j < len(a):
                right = a[j]
                h = 0
                char_num = str(cnall[left])
                if len(char_num) == 1:
                    char_num = str(h) + char_num
                # print(charNum)
                char_num1 = str(cnall[right])
                if len(char_num1) == 1:
                    char_num1 = str(h) + char_num1
                # print(charNum1)
                d = int(str(char_num) + str(char_num1))
                z = (d - int(shift_1))
                y = (int(minverse_) * z)
                crypt = (y % 5152)
                crypt_char = str(crypt)
                print(crypt_char)
                if len(crypt_char) == 3:
                    crypt_char = str(h) + crypt_char
                elif len(crypt_char) == 2:
                    crypt_char = str(h) + str(h) + crypt_char
                print(crypt_char)
                crypt_Text.append(crypt_char)

                break

            break
    return ''.join(crypt_Text)


def convert52(crypt_text):
    affine = []

    for char in map(operator.add, crypt_text[::2], crypt_text[1::2]):
        crypted1 = str(char)
        s = 0
        t = 1

        while s < len(crypted1):
            h = 0
            left = crypted1[s]
            if left == str(h):
                left = ''

            while t < len(crypted1):
                right = crypted1[t]
```

```

                combine = (str(left) + str(right))
                print(combine)
                lookup = ncall[int(combine)]
                affine.append(lookup)
                break
            break
    return ''.join(affine)


while True:

    with open('F:\kazmi\Vigenere-and-block-affine-cipher-master/plaintext.txt', '
    r') as myfile:
        affineencrypt = myfile.read().replace('\n', '')
    print(affineencrypt)

    shift = input("Enter value of b\n\n")
    system('cls')

    multiplier = input("Enter value of m\n\n")
    system('cls')

    choice = input("Enter S or L\n\n")
    system('cls')

    if choice.lower() == 's':
        modulus = 2526
        minverse = modinv(multiplier, modulus)
        affineencrypttext = affineencrypt.upper()
        cryptText = crypted26(affineencrypttext, shift, minverse)
        print(cryptText)
        AffinedecryptText = convert26(cryptText)
        print(AffinedecryptText)
        print('done')
        cryptDir = open('F:\kazmi\Vigenere-and-block-affine-cipher-master/
    plaintext.txt', 'w')
        cryptDir.write(cryptText)
        cryptDir.close()
        break

    elif choice.lower() == 'l':
        modulus = 5152
        minverse = modinv(multiplier, modulus)
        cryptText = crypted52(affineencrypt, shift, minverse)
        print(affineencrypt)
        AffinedecryptText = convert52(cryptText)
        print(AffinedecryptText)
        print('done')
        cryptDir = open('F:\kazmi\Vigenere-and-block-affine-cipher-master/
    plaintext.txt', 'w')
        cryptDir.write(cryptText)
        cryptDir.close()
        break
    break
end = time. time()
print(end - start)
```

Journal publications

1 Kazmi, H.S.Z., Javaid, N., Awais, M., Tahir, M., Shim, S., Zikria, Y.B. **2019**. "Congestion Avoidance and Fault Detection in WSNs using Data Science Techniques". In Transactions on Emerging Telecommunications Technologies. ISSN: 2161-3915.

2 Zahid, M., Ahmed, F., Javaid, N., Abbasi, R.A, Kazmi, H.S.Z, Javaid, A., Bilal, M., Akbar, M., Ilahi, M. **2019**. "Electricity Price and Load Forecasting using Enhanced Convolutional Neural Network and Enhanced Support Vector Regression in Smart Grids." Electronics, 8(2), 122, EISSN 2079-9292.

Conference proceedings

1 Kazmi, H.S.Z., Nazeer, F., Mubarak, S., Hameed, S., Basharat, A., Javaid, N. **2019**. "Trusted Remote Patient Monitoring using Blockchain-based Smart Contracts.". In 14-th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA).

2 Kazmi, H.S.Z., Javaid, N., Imran, M., Outay, F. **2019** "Congestion Control in Wireless Sensor Networks based on Support Vector Machine, Grey Wolf Optimization and Differential Evolution.". In 11th Wireless Days Conference (WD).