UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

Electricity Theft Detection in Smart Grids Based on Deep Neural Network

by

Leloko James Lepolesa

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

June 2022

Abstract

Electricity theft is a global problem that negatively affects both utility companies and electricity users. It destabilizes the economic development of utility companies, causes electric hazards and impacts the high cost of energy for users. Development of and adherence to smart grids play an important role in the development of theft detection measures. Smart grids generate massive data that includes customer consumption data which, through machine learning and deep learning techniques, can be utilized to detect electricity theft. In this work, statistical analyses are undertaken to investigate the difference in consumption patterns between faithful and unfaithful electricity users. Dataset weaknesses such as missing data and class imbalance problems are addressed through data interpolation and synthetic data generation processes. Comprehensive features in time and frequency domains are extracted and used in a fully connected feed-forward deep neural network classifier. The minimum redundancy maximum relevance scheme is used to analyse individual features' contribution to successful classification, thereby validating frequency-domain features' dominance over time-domain features. The principal component analysis is employed to reduce the dimensionality of the classifier input while keeping the results satisfactory to simplify the training process. Electricity theft detection performance is improved by optimizing hyperparameters using a Bayesian optimizer. An adaptive moment estimation optimizer is employed to carry out experiments using different values of key parameters to determine the optimal settings that achieve the best accuracy. The classifier achieves 97% area under the curve (AUC), which is 1% higher than the best AUC in existing works evaluated on the same dataset, and 91.8% accuracy, which is the second-best on the benchmark.

Acknowledgements

I would like to thank the Almighty God for granting permission and making it possible to do this work. Special thanks are given to my supervisor Prof. Ling Cheng and co-supervisor Mr Shamin Achari for their dedicated support and guidance throughout the course of this work. I would like to thank my family for their continued encouragement and support.

I would like to acknowledge the Center for Telecommunications Access and Services (CeTAS) at the Wits School of Electrical and Information Engineering and the South African National Research Foundation (NRF) for financial assistance.

Table of Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

AdaGrad	**Ada**ptive **Grad**ient Algorithm
Adam	**Ada**ptive **M**oment Estimation
AMI	**A**dvanced **M**etering **I**nfrastructure
ANN	**A**rtificial **N**eural **N**etwork
AUC-ROC	**A**rea **U**nder the **C**urve of **R**eceiver **O**perator **C**haracteristic Curve
BGD	**B**atch **G**radient **D**escent
CNN	**C**onvolutional **N**eural **N**etworks
CS-SVM	**C**ost-**S**ensitive **S**upport **V**ector **M**achine
CWGAN-GP	**C**onditional **W**asserstein **G**enerative **A**dversarial **N**etwork with **G**radient **P**enalty
DNN	**D**eep **N**eural **N**etwork
ENN	**E**dited **N**earest **N**eighbour
FN	**F**alse **N**egatives
FP	**F**alse **P**ositives
GSM	**G**lobal **S**ystem for **M**obile Communications

IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
KPCA	**K**ernel Function and **P**rincipal **C**omponent **A**nalysis
LGB	**L**ight **G**radient **B**oosting
LOF	**L**ocal **O**utlier **F**actor
LSTM	**L**ong **S**hort-**T**erm **M**emory
MAE	**M**ean **A**bsolute **E**rror
MbGD	**M**ini-**b**atch **G**radient **D**escent
MCC	**M**atthews **C**orrelation **C**oefficient
MCU	**M**icro**c**ontroller **U**nit
mRMR	**M**inimum **R**edundancy **M**aximum **R**elevance
MSE	**M**ean **S**quared **E**rror
NTL	**N**on-**T**echnical **L**oss
O-SVM	**O**ne-Class **S**upport **V**ector **M**achine
OPF	**O**ptimum **P**ath **F**orest
PCA	**P**rincipal **C**omponent **A**nalysis
PC	**P**rincipal **C**omponent
PPV	**P**ositive **P**redictive **V**alue
ReLU	**Re**ctified **L**inear **U**nit
RFID	**R**adio **F**requency **Id**entification
RMSE	**R**oot **M**ean **S**quared **E**rror
RMSProp	**R**oot **M**ean **S**quared **Prop**agation

ROI	**R**eturn **o**n **I**nvestment
SALM	**S**MOTEENN**A**lexNet-**L**GB **M**odel
SGCC	**S**tate **G**rid **C**orporation of **C**hina
SGD	**S**tochastic **G**radient **D**escent
SMOTE	**S**ynthetic **M**inority **O**ver-sampling **Te**chnique
SMS	**S**hort **M**essage **S**ervice
SVD	**S**ingular **V**alue **D**ecomposition
SVM	**S**upport **V**ector **M**achine
TL	**T**echnical **L**oss
TN	**T**rue **N**egatives
TPR	**T**rue **P**ositive **R**ate
TP	**T**rue **P**ositives

List of Symbols

ϵ	AdaGrad's smoothing term
η	Learning rate
$\eta_i^{(t)}$	Learning rate for the i^{th} parameter at a given iteration step
λ	Eigenvalue
$\mathbf{C_X}$	Covariance matrix of matrix of input features
MI	Mutual information
$\mathbf{W}$	Weights matrix
$\mathbf{w}$	Weights vector
$\mathbf{w}_h$	Hidden neurons' weights vector
$\mathbf{w}_i$	Input weights vector
$\mathbf{w}_o$	Output weights vector
$\mathbf{X}$	Matrix of input features
$\mathbf{x}$	Input vector of a deep neural network model
$\mathbf{y}$	Output vector of a deep neural network model
$\mathbf{y}_i^{(a)}$	Actual value of the model's output at the i^{th} position
$\mathbf{y}_i^{(p)}$	Predicted value of the model's output at the i^{th} position
θ	Neural network model parameters
$\theta_i^{(t)}$	Model's i^{th} parameter at a given iteration step
B	Batch

b	Neuron's bias
c	Principal component
e	Euler's number
g	Output of a neuron
L	Loss function
$m_i^{(t)}$	The i^{th} gradient of the loss function at the given iteration step
R_d	Minimum redundancy
R_l	Maximum relevance
T	Training dataset
t	Iteration step
w_i	The i^{th} value in the weights vector
x_i	The i^{th} value in the input vector
y_m	The m^{th} output of a model
z	Input of a neuron's activation function
$\nabla_\theta L(\theta^{t-1}; T)$	Gradient of a loss function

Chapter 1

Introduction

1.1 Introduction

Electricity is an essential need in modern society. The growing population and continually increasing urbanization effect on increasing electricity demand, especially for thermal comfort as the need for heating and air conditioning increases [1]. Power grids usually consist of multiple power generation entities and operators who employ different forms of communication and coordination [2]. Smart grids are the type of power grids designed with the aim to deliver the highest possible quality electricity to customers at the lowest cost [3]. They bring forth better connectivity, automation and coordination between power generation, distribution and grid management entities [2].

Smart grids are usually composed of traditional power grids, smart meters and sensors, computing facilities to monitor and control grids, e.t.c, all connected through the communication network [4]. Smart meters and sensors collect data such as electricity usage, grid status, electricity price, etc [4]. Smart grids suffer electricity losses categorized into technical losses (TLs) and non-technical losses (NTLs). TLs are caused by the properties of power grid components [5]. Examples include power dissipation through internal resistances of transmission lines, power transformers and measurement systems [5].

NTLs are external to the power grid properties [6]. More than $96 billion is lost

by utility companies around the world every year due to NTLs [7]. Examples include electricity theft, faulty meters and billing errors [6], of which electricity theft is the major contributor [7]. Electricity theft is a problem that affects utility companies worldwide. In sub-Saharan Africa, 50% of generated energy is stolen, as reported by World Bank [8]. [9] reports that in 2015, India lost $16.2 billion, Brazil lost $10.5 billion and Russia lost $5.1 billion. It is estimated that approximately $1.31 billion (R20 billion) revenue loss incurred by South Africa (through Eskom) per year is due to electricity theft [8].

Apart from revenue loss, electricity theft has a direct negative impact on the stability and reliability of power grids [10]. It can lead to surging electricity, electrical systems overload and public safety threats such as electric shocks [11]. It also has a direct impact on energy tariff increases which affect all customers [10]. Implementation of smart grids comes with many opportunities to solve the electricity theft problem [11].

Many Utilities sought to curb electricity theft in traditional grids by examining meters' installation and configurations, checking whether the power line is bypassed, etc. [11]. These methods are expensive, inefficient and cannot detect cyber attacks [11], [12]. Recently, researchers have worked towards detecting electricity theft by utilizing machine learning classification techniques using readily available smart meters data. These theft detection methods have been proved to be of relatively lower costs [13]. However, currently existing classification techniques concentrate much on time-domain features and neglect frequency domain features, thereby limiting their performance.

Regardless of the fact that there is active ongoing research on electricity theft detection, electricity theft is still a problem. The major cause of delay in solving this problem may be that smart grids deployment is realized in developed nations while developing nations are lagging behind [14]. The challenges of deploying smart grids include the lack of communication infrastructure and users' privacy concerns over data reported by the smart meters [15]. However, [15] reports that smart meters are being considered by many developed and developing countries with aims that include solving NTLs. [16] predicted smart grids global market to triple in size between 2017 and 2023, with the following key regions leading smart grids deployment: North America, Europe and Asia.

This work presents a novel electricity theft detection scheme based on carefully extracted and selected time-domain and frequency-domain features in a deep neural network (DNN)-based classification approach. The results show that employing frequency-domain features as opposed to using time-domain features alone enhances classification performance. Principal component analysis (PCA) is used for dimensionality reduction to simplify the training process and each of the features' contribution to classification performance is analyzed using minimum redundancy maximum relevance (mRMR) scheme. A realistic electricity consumption dataset released by State Grid Corporation of China (SGCC) accessible at [17] was used. The dataset consists of electricity consumption data taken from January 2014 to October 2016.

1.2 Problem Statement

Due to advanced metering infrastructure (AMI) in smart grids, attack surface which should be protected by utility companies is broadened by the introduction of cyber threats on physically accessible smart grid devices [18], [19]. The attacks on smart grids may be physically or cyber oriented. Physical attacks include tapping external sources such as distribution feeders, tempering with smart meter wirings, and applying magnets to interfere with electromechanical rotors or solid-state current transformers [18].

Cyber-attacks include tempering communication channels by jamming smart meter wireless communications and accessing and modifying smart meters' firmware and storage [20]. All these attacks cause electricity consumption data collected from the smart grids to falsely reflect the actual consumption of electricity by customers.

To solve the electricity theft problem, this research aims to answer the following question:

What level of performance can be realized by using time and frequency domains features to detect electricity theft in a DNN-based classification approach? An investigation is done to determine how mRMR can help in identifying individual features' contribution to successful theft detection, as well as how PCA can be

used to reduce DNN input dimensionality to simplify the training process while maintaining electricity theft detection performance satisfactory.

1.3 Research Motivation and Significance

The electricity theft problem negatively affects the worldwide economy. Traditional methods that are used to detect electricity theft are expensive and less effective. To reduce costs and improve the effectiveness of electricity theft detection measures, the electricity consumption characteristics of faithful and unfaithful customers should be clearly differentiated. This can be achieved by employing machine learning classification techniques to work on available smart meters data.

The significance of this research is as follows:

1. This research uses comprehensive time-domain features in a DNN based classification approach and shows how using frequency-domain features enhances classification performance.

2. To the best of the author's knowledge, this is the first work to validate the importance of frequency-domain features over time-domain features to detect electricity theft.

3. The PCA is used to perform a classification task with reduced features dimension. The results are compared with results obtained from classification done with all input features to interpret the results and simplify the future training process.

4. The mRMR scheme is used to identify the most significant features and to validate the importance of frequency-domain features over time-domain features for detecting electricity theft.

5. Lastly, the hyperparameters of the model are optimized for overall improved performance using a Bayesian optimizer. An adaptive moment estimation (Adam) optimizer is used to determine the best ranges of values of the key parameters that can be used to achieve good results with optimal model training speed.

1.4 Objectives and Scope

This project aims to detect electricity theft using DNN classifier. The objectives are as follows:

- Existing electricity theft detection methods are studied to determine their limitations.
- Electricity consumption data is acquired from [17]. It is a realistic electricity consumption dataset released by the State Grid Corporation of China (SGCC). It contains faithful and unfaithful customers' electricity consumption data samples taken from January 2014 to October 2016.
- Statistical analyses are made on the dataset to determine its quality and improve it to minimize possible detection errors caused by poor dataset quality. Customers' load profile is also analysed to determine the difference between faithful and unfaithful customers' consumption behaviour so that distinguishing features can be decided.
- Comprehensive time-domain and frequency-domain features are extracted and fed to a rule of thumb methods' designed feed-forward DNN classifier.
- The classifier is repeatedly trained with time-domain, frequency-domain and all features from both domains combined to analyse the classification performance.
- MRMR feature selection scheme is used to analyse each feature's contribution to successful classification.
- PCA is used to reduce DNN input dimensionality to simplify the training process without compromising classifier performance.
- Using adaptive moment estimation (Adam) optimizer, significant parameters are studied to determine the range of values that give the best classification performance in terms of accuracy and training time.
- Hyperparameters are tuned using a Bayesian optimizer to further improve the classifier's performance.

This work makes use of a labelled dataset that contains historical electricity consumption data of faithful and unfaithful customers without exposing their identities, therefore there is no ethical clearance required. Dataset does not include the kind of theft committed by each unfaithful customer, therefore this work does not take into account the type of theft committed, but detects theft based on data consumption pattern only.

1.5 Dissertation Organisation

The rest of this dissertation is organised as follows:

Chapter 2 gives the background of the techniques used in this research. It covers the DNN details, with much concentration on the fully connected feed-forward DNN. It also covers dimensionality reduction techniques used in this work, which are PCA and mRMR. Chapter 3 presents an electricity theft detection scheme. This chapter is based on the research paper produced from this work. In Chapter 4, a research summary is given, recommendations for future directions and a conclusion is drawn. The conclusion answers the research question based on the results obtained.

1.6 Chapter Summary

This chapter introduced the background of smart grids and associated TLs and NTLs, with special emphasis on electricity theft. The slow deployment of the smart grids has been pointed out as the possible reason why electricity theft still remains the main challenge in NTLs. The problem statement looked into different attacks on the smart grid, which, thankfully, have an impact on the consumption data of customers. This led to the formulation of the research question as well as the significance of this research. The next chapter gives the background of techniques used in this research.

Chapter 2

Background

2.1 Introduction

This chapter gives the background of techniques employed in this research to detect electricity theft. It covers the concept of DNNs, particularly fully connected feed-forward DNNs. Their history of development, training and optimization are covered. Dimensionality reduction is also presented, with emphasis on PCA and mRMR feature selection scheme.

2.2 Deep Neural Networks

Artificial neural networks (ANNs) are a class of machine learning techniques that have been built to imitate biological human brain mechanisms [21], [22]. They are typically used for extracting patterns or detecting trends that are difficult to be detected by other machine learning techniques [23].

They consist of multiple layers of nodes/neurons which are connected to subsequent layers [22]. A neuron is the basic element of a neural network, which originates from the McCulloch-Pitts neuron, a simplified model of a human brain's neuron [24]. Figure 2.1 shows a model diagram of a neuron that comprises a layer following the input to the ANN.

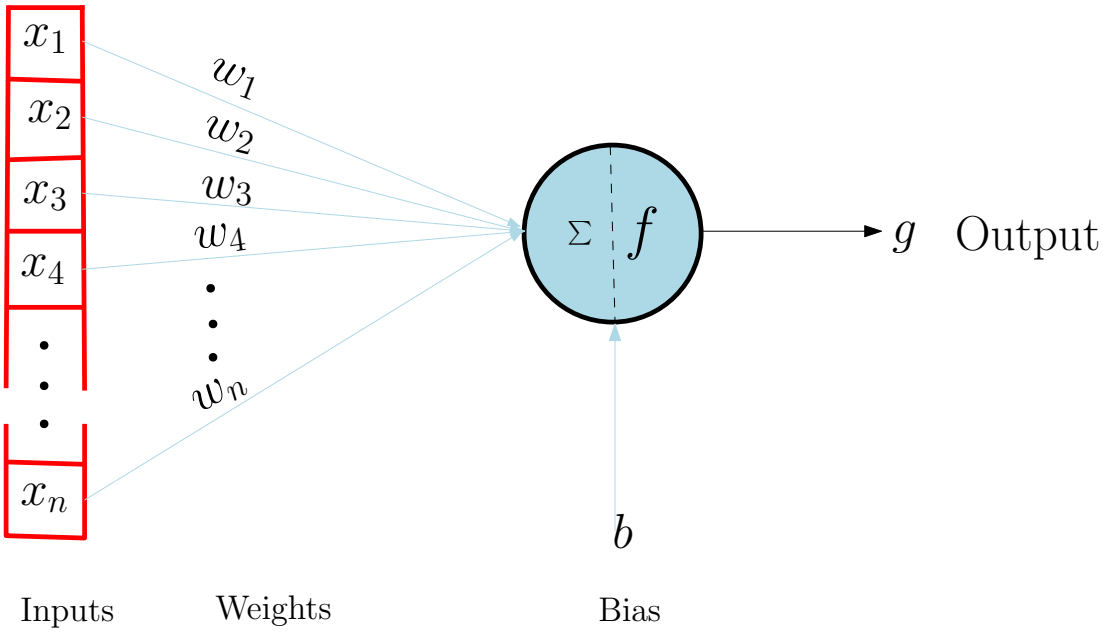


Figure 2.1. First hidden layer neuron model

It consists of an activation function f, which takes a weighted sum of the real number input signal and gives real number output y given by Equation (2.1):

$$g = f(\sum (w_i x_i) + b), \tag{2.1}$$

where $x_i \in \mathbf{x}$, $w_i \in \mathbf{w}$, $\mathbf{x}$ is input vector, $\mathbf{w}$ is weights vector and b is the bias [24]. Neural network nodes mimic the brain's neurons, while connection weights mimic connections between neurons, which are unique for each connection [21], [22]. A neural network stores information in the form of weights and biases. A neural network stores information in the form of weights $\mathbf{w}$ and bias b.

The deep neural networks (DNNs) concept originates from research on ANNs [25]. DNNs are characterized by two or more hidden layers [21]. They are able to learn more complex and abstract features than shallow ANNs [26]. Oftentimes in classification problems, the output layer is made up in such a way that one neuron represents a certain class [22]. All neural network layers are used to filter and learn the complicated features, except for an output layer which classifies based on learnt features [22], [27]. Before DNNs development, most machine learning techniques explored architectures of shallow structures which commonly contain

a single layer of non-linear transformation [25]. Examples of these architectures include support vector machines (SVMs), logistic regression and ANNs with one hidden layer.

DNNs have different architectures, which are used to solve different problems. Examples of DNN architectures include feed-forward DNN, convolutional DNN and recurrent DNN. In this research work, a fully connected feed-forward DNN was used. the typical structure of a fully connected feed-forward DNN is shown in Figure 2.2.

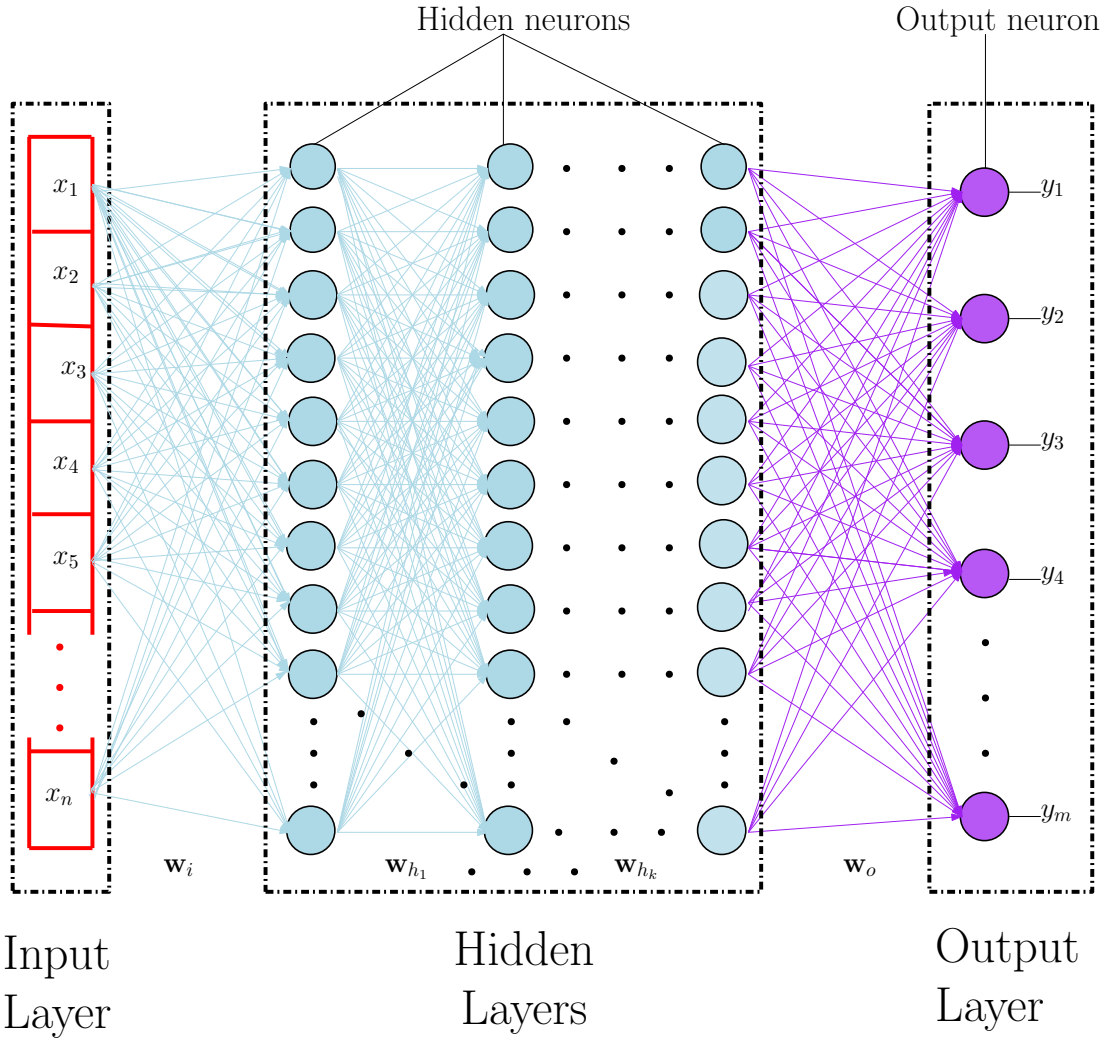


Figure 2.2. Fully connected feed-forward DNN general architecture

The DNN given in Figure has the following major parts:

Input Layer (x)

A layer that comprises input data features or representation.

Input Weights ($\mathbf{w}_i$)

Weights of the connections between the input layer and the first hidden layer of a DNN.

Hidden Layers

The layers of neurons between the input and output layers. They are used to analyse the relationship between the input and output signals [23].

Hidden Neurons Weights ($[\mathbf{w}_{h_1}, \cdots, \mathbf{w}_{h_k}]$)

Weights of the connections between the hidden layers.

Output Weights ($\mathbf{w}_o$)

Weights between the last hidden layer and the output layer.

Output Layer (y)

The last layer of a DNN. It gives the output of the network from network inputs.

In a feed-forward architecture, computation is a sequence of operations on the output of a previous layer. The final operations generate the output. For a given input, the output stays the same, it does not depend on the previous network input [26].

2.2.1 History of DNNs Development

[26] reports that ANNs were first proposed in the 1940s, and research on DNNs emerged in the 1960s. In 1989, the LeNet network, which used many digital neurons, was built for recognizing hand-written digits. Major breakthroughs were seen in the years beyond 2010, with examples such as Microsoft's speech recognition system, AlexNet image recognition system, and DNN accelerator research such as Neuflow and DianNao brought into play.

The following reasons are reported by [23], [25], [26] as major contributors to DNNs' improved development:

- Advancements in semiconductor devices and computer architecture, leading to parallel computing and lower costs of computer hardware.
- Huge amount of data obtained by cloud providers and other businesses, making large datasets that train DNNs effectively.
- Advances in machine learning and signal/information processing research which leads to the evolution of techniques to improve accuracy and broaden the domain of DNNs application.

With current technology, it is possible to build DNNs that have more than a thousand layers [26].

2.2.2 DNN Training

A large dataset and high computational abilities are the major requirements in training the DNN since weight updates require multiple iterations [26]. DNN training process is concerned with adjusting the weights between the neurons [23]. Through the training process, the DNN learns information from the data. Learning can be in the following major four ways: supervised, semi-supervised, unsupervised or reinforcement.

Supervised Learning

In a supervised learning manner, the network is trained with the labelled dataset [26], [28], [29]. The training data is passed through the network for a number of iterations (or epochs) until the loss is below the reasonably accepted threshold. Supervised learning is very common in training neural networks [28], [29].

Semi-supervised Learning

In a semi-supervised learning manner, part of the training dataset is labelled [26]. The training procedure is still the same as training with supervised learning.

Unsupervised Learning

In unsupervised learning, all training dataset samples are not labelled. It is typically used to find the structure of clusters underlying the data, and the clusters are discovered during training [26], [28], [29].

Reinforcement Learning

With reinforcement learning, the training dataset is unlabelled. However, the network is rewarded for each prediction (i.e. it is told whether the prediction is correct or not correct) [28], [29].

In this research, supervised learning was used. The typical procedure for supervised learning in DNNs as given by [21], [27] is as follows:

1. The weights $\mathbf{W} = [\mathbf{w}_i, \mathbf{w}_{h_1}, \cdots, \mathbf{w}_{h_k}, \mathbf{w}_o]$ are initialized with adequate values.
2. Input signal $\mathbf{x}$ is fed to the network's input layer.
3. Output error is calculated, and then weights are adjusted with an aim to reduce an error.

4. Steps 2 and 3 are repeated for all training data.

Backpropagation

A loss function of a multi-layered ANN is composed of weights from successive layers between input and output layers [29]. Backpropagation uses the chain rule to obtain the gradient of the loss function in terms of summation of local gradient products over different nodes connections between input and output layers [21], [22], [29]. Backpropagation algorithms typically use gradient-based optimization algorithms to update the neural network parameters on each layer [30].

Backpropagation can be divided into the forward and backward phases as the two main phases [29]. They are described as thus:

- **Forward Phase**

  During the forward phase, the training data is fed to an ANN and activations are computed with current parameters. The predicted output is compared with the expected output and the loss is determined. The gradient of the loss function with respect to the output is then determined.

- **Backward Phase**

  After determining the gradient of the loss function with respect to the output, the chain rule is used to compute the gradient of the loss function with respect to the hidden layers. The gradient is computed from the output layer backwards [29].

Activation Function

An activation function takes an input signal. By simulating a response of a biological neuron, it transforms an input signal into an output signal which may be an input to another neuron [31], [32]. There are many activation functions, which can be generally divided into two kinds: the linear and non-linear activation functions. The type of activation function used in a DNN plays a major role in the prediction accuracy of the model [32]. The selection of an activation

function depends on the reasons such as computational power, analytic flexibility and whether the desired output should be continuous or discrete [23]. Let $z = \sum (w_i x_i) + b$. Then Equation (2.1) can be re-written as shown in Equation (2.2):

$$g = f(z). \tag{2.2}$$

Linear Activation Functions

Linear activation functions usually have an activation that is directly proportional to the input. They can be expressed in the form of Equation (2.3):

$$f(z) = Cz, \tag{2.3}$$

where C is a constant. The output of the linear activation function is in the range $(-\infty, \infty)$ and its derivative is $f^{'}(z) = C$. Since the gradient is not related to the input, an error can not be minimized by the use of a gradient [33]. This activation function is normally used in regression problems [34].

Non-linear Activation Functions

Non-linear activation functions are widely used in DNNs because of their ability to adapt to data variations and differentiate outputs [33]. Among the many developed non-linear activation functions, the most popular are described as follows [31]–[34].

- **Sigmoid Activation Function**

  Sigmoid activation function is given by Equation (2.4):

  $$f(z) = \frac{1}{1 + e^{-z}}, \tag{2.4}$$

  where input $z \in (-\infty, \infty)$ and an activation $f \in (0, 1)$.

  Sigmoid is continuous and differentiable everywhere. Its derivative is given by Equation (2.5):

$$f^{'}(z) = \frac{e^{-z}}{(1+e^{-z})^2}. \tag{2.5}$$

Due to less computation in finding its derivative, this activation function is widely used in shallow neural networks. It is rarely used in DNNs' hidden layers because of its soft saturation property which makes DNNs delay converging during training.

- **Hyperbolic Tangent Activation Function**

  Like the sigmoid, hyperbolic tangent is continuous and differentiable everywhere. It is given by Equation (2.6), and its derivative is given by Equation (2.7):

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{2.6}$$

$$f^{'}(z) = \frac{4e^{-2z}}{(1+e^{-2z})^2}. \tag{2.7}$$

  The input $z \in (-\infty, \infty)$ and an activation $f \in (-1, 1)$. Using a hyperbolic tangent for activation makes the neural networks converge faster than when using a sigmoid, therefore a hyperbolic tangent is more preferred than a sigmoid.

- **Rectified Linear Unit Activation Function**

  Rectified linear unit (ReLU) activation function is given by Equation (2.8), and its derivative by Equation (2.9):

$$f(z) = \max(0, z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}, \tag{2.8}$$

$$f^{'}(z) = \max(0, z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}. \tag{2.9}$$

  Compared to sigmoid and hyperbolic tangent activation functions, ReLU is the simplest and most commonly used in DNNs because of its good property

of being close to linear, hence better convergence. It is more efficient since it activates less number of neurons at the same time. For $z > 0$, its gradient is constant thereby avoiding the vanishing gradient problem. Its gradient is cheaper to compute as there are no calculations that involve exponents.

- **Softmax Activation Function**

  Softmax activation function is given by Equation (2.10):

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.10}$$

  where K is the number of classes.

  Softmax is typically used in the output layer of a DNN for classification purposes. The output of a softmax is a probability of a particular class j, therefore if the softmax activation function is used in the output layer, all of the output layer activations sum to 1.

2.2.3 DNN Optimization

Optimization is done to define a function that makes the solution through DNN networks easier by finding the values of the parameters that minimize an objective (loss) function [35]. Many loss functions are used to measure the model losses. For the model output $\mathbf{y}$ consisting of m values, let $\mathbf{y}_i^{(a)}$ and $\mathbf{y}_i^{(p)}$ be actual and predicted values of $\mathbf{y}$ respectively at the i^{th} position. The popularly used loss functions are given below [28], [29], [36]:

- Mean Squared Error (MSE)

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{y}_i^{(a)} - \mathbf{y}_i^{(p)})^2. \tag{2.11}$$

- Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\mathbf{y}_i^{(a)} - \mathbf{y}_i^{(p)})^2}. \tag{2.12}$$

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|\mathbf{y}_i^{(a)} - \mathbf{y}_i^{(p)}|. \tag{2.13}$$

- Cross-entropy Loss H

$$H = -\sum_{i=1}^{m}\mathbf{y}_i^{(a)}\log(\mathbf{y}_i^{(p)}). \tag{2.14}$$

  For binary cross-entropy loss, $m = 2$. In this work, binary cross-entropy loss is utilized. Here $\mathbf{y}_i^{(a)}$ takes 0 for faithful customers and takes 1 for unfaithful customers, and $\mathbf{y}_i^{(p)}$ is the softmax probability for the i^{th} class.

MSE, RMSE and MAE are typically used in regression problems, while cross-entropy loss is commonly used for classification problems [30]. Optimization can be done through optimization algorithms that can be roughly classified into the following three classes: derivative-free, first order and high order optimizers. Adam [37], a first-order and a type of gradient descent optimization method, and Bayesian optimization, which is derivative-free, were used in this research. They are introduced as follows.

Adaptive Moment Estimation

Adam is a gradient descent optimization algorithm that was introduced by [37]. To dive into Adam's details, gradient descent is introduced as follows.

Gradient descent is the most popular first-order iterative optimization algorithm used to train a DNN [30], [35], [38]. With gradient descent, parameters are updated continually on each iteration until the model converges, and the final values are regarded as the optimal parameters [30]. It has the following three variant algorithms which are continuously developed to improve its performance: batch gradient descent (BGD), stochastic gradient descent (SGD) and mini-batch gradient descent (MbGD).

The variants are characterized by the portion of data from the training dataset

they require to compute the gradient of the loss function [38]. These three variants serve as the base algorithms to recently developed gradient descent algorithms, including Adam [30].

- **Batch Gradient Descent**

  With BGD (also known as vanilla gradient descent), the gradient of the loss function is computed for the whole dataset to perform parameters update. The update criterion is given by Equation (2.15):

  $$\theta^{(t)} = \theta^{(t-1)} - \eta\dot{\nabla}_\theta L(\theta^{(t-1)}; T), \tag{2.15}$$

  where θ is the set of parameters updated, t is updating iteration step, η is the learning rate L is the loss function and T is the training dataset. The term $\nabla_\theta L(\theta^{t-1}; T)$ is the gradient of the loss function L. At the training initialization (i.e., when $t = 0$), the initial parameters $\theta^{(0)}$ are initialized subject to certain distributions such as uniform distribution and normal distribution [35].

  BGD has the following benefits: stable convergence and balanced error gradient [35]. Its weaknesses are: it demands the whole training dataset be in memory and it usually converges to the local minimum closer to parameters initial placement.

- **Stochastic Gradient Descent**

  With SGD, a loss function gradient is computed and the update of parameters is made for each training example $\mathbf{x}_i$. A stochastic mechanism/method is one whereby execution is done based on a random possibility [35]. In SGD, samples are typically randomly chosen by shuffling the training dataset before the training process [35], [38]. SGD update criterion is given by Equation (2.16):

  $$\theta^{(t)} = \theta^{(t-1)} - \eta\dot{\nabla}_\theta L(\theta^{(t-1)}; (\mathbf{x}_i, \mathbf{y}_i)), \tag{2.16}$$

  where $(\mathbf{x}_i, \mathbf{y}_i) \in T$.

SGD has the following benefits: updating parameters after every sample increases the chances of finding the global minimum, it is faster than BGD as it does not recalculate gradients for similar examples before each update of the parameters. Its weaknesses are: though it is faster than BGD, it may cause heavy fluctuations on parameter updates and on the loss function, and by setting a small value of learning rate and decreasing it during the training process, SGD may converge to the local minimum like BGD.

- **Mini-batch Gradient Descent**

  With MbGD, loss function gradient is computed for training examples of size B. MbGD update criteria is given by Equation (2.17):

$$\theta^{(t)} = \theta^{(t-1)} - \eta\dot{\nabla}_{\theta}L(\theta^{(t-1)}; B), \tag{2.17}$$

  where $B \subset T$.

  Similarly to SGD, samples are typically randomly chosen by shuffling the training dataset before the training process [35], [38].

  MbGD is faster compared to BGD, it has a more stable convergence compared to SGD and it is widely used to train DNNs.

Gradient descent is more efficient and practical to DNNs than higher-order derivative-based algorithms [30]. Many gradient descent algorithms have been developed to optimize neural networks. Adam was developed to improve the performance of its predecessor algorithms adaptive gradient algorithm (AdaGrad) [39] and root mean squared propagation (RMSProp). Both AdaGrad and RMSProp are based on MbGD. They are introduced as follows.

- **AdaGrad**

  AdaGrad uses a different learning rate for updating each parameter $\theta_i^{(t)}$ at a given iteration step t. A learning rate $\eta_i^{(t)}$ for the parameter $\theta_i^{(t)}$ at a given iteration step t is given by Equation (2.18):

$$\eta_i^{(t)} = \frac{\eta}{\sqrt{\mathbf{M}_i^{(t)} + \epsilon}}, \tag{2.18}$$

where $\mathbf{M}_i^{(t)}$ is a diagonal matrix where each element in the diagonal is the sum of squares of the gradients with respect to θ_i until iteration t. ϵ is a smoothing term to avoid division by zero. It is typically set to $\epsilon = 10^{-8}$.

Let $m_i^{(t)} = \nabla_\theta L(\theta_i^{(t-1)}; B)$ denote the i^{th} gradient at an iteration t. Then $\theta_i^{(t)}$ is updated with Equation (2.19):

$$\theta_i^{(t)} = \theta_i^{(t-1)} - \eta_i^{(t)} m_i^{(t)}. \tag{2.19}$$

The benefit of AdaGrad is that there is no need to manually tune the learning rate. However, the weakness is that the learning rate keeps decreasing every time gradients are squared in the denominator during the training process. After many iterations, the network no longer learns as the learning rate becomes too small to update the parameters.

- **RMSProp**

  RMSProp was proposed by Geoff Hinton in his Coursera class and he did not publish it [38]. It was developed to resolve the problem of decaying gradient in AdaGrad. Unlike AdaGrad which uses all the past gradients to compute the learning rate for each parameter, RMSProp accumulates only gradients of limited window size. However, gradients are not stored, but the sum of gradients is recursively defined as a decaying average of all past gradients.

  At a given iteration t, the running average $A(m^2)^{(t)}$ is given by Equation (2.20):

$$A(m^2)^{(t)} = \gamma A(m^2)^{(t-1)} + (1-\gamma) m^{2(t)}, \tag{2.20}$$

  where γ is typically set to 0.9.

  The changing learning rate is then updated with Equation (2.21):

$$\eta_i^{(t)} = \frac{\eta}{\sqrt{A(m^2)^{(t)} + \epsilon}}. \tag{2.21}$$

Just like AdaGrad and RMSProp, Adam computes the adaptive learning rate for each parameter θ_i. It stores exponentially decaying average $v^{(t)}$ of past squared

gradients like RMSProp as shown by Equation (2.22), and in addition to that, it keeps exponentially decaying average of past gradients as shown by Equation (2.23):

$$v^{(t)} = \beta_2 v^{(t-1)} + (1 - \beta_2) m^{2(t)}, \tag{2.22}$$

$$u^{(t)} = \beta_1 u^{(t-1)} + (1 - \beta_1) m^{(t)}. \tag{2.23}$$

To prevent $u^{(t)}$ and $v^{(t)}$ from being biased towards zero during early iteration steps after initialization, $\tilde{u}^{(t)}$ and $\tilde{v}^{(t)}$ are defined as follows:

$$\tilde{u}^{(t)} = \frac{u^{(t)}}{1 - \beta_1^{(t)}}, \tag{2.24}$$

$$\tilde{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^{(t)}}. \tag{2.25}$$

Adam parameters update rule is then computed by Equation (2.26):

$$\theta^{(t)} = \theta^{(t-1)} - \frac{\eta}{\sqrt{\tilde{v}^{(t)}} + \epsilon} \tilde{u}^{(t)}. \tag{2.26}$$

The recommended values of β_1, β_2 and ϵ are 0.9, 0.999 and 10^{-8} respectively.

Bayesian Optimization

Bayesian optimization uses Bayes' formula to find the posterior information about an objective function distribution based on the sample information and the prior information about the distribution of an objective function [40]–[42]. It can be used to effectively solve functions that do not have the closed-form expression, functions that are hard/expensive to evaluate, and non-convex functions [40], [41]. Due to this property, Bayesian optimization is widely employed to optimize

hyperparameters in DNNs [41], [42].

Tuning hyperparameters can be seen as an optimization problem where an objective function is unknown. This makes it hard to use gradient descent methods and other derivative-based optimization methods [40]. It deals with finding the best combination of hyperparameters that achieves the best performance on data in a reasonable amount of time [40].

[40] classifies hyperparameters optimization into the manual search and automatic search kinds which are defined as follows:

- **Manual search** is achieved by tuning hyperparameters by hand. It is done based on the user's intuition and requires substantial experience for the model to perform satisfactorily.
- **Automatic search** is achieved by employing a computer-defined optimization algorithm to tune hyperparameters. Examples of the widely used algorithms include grid search, random search, and Bayesian optimization algorithm.

Bayesian optimization is derived from Bayes' theorem which states that for events A and B,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.27}$$

This optimization method determines the distribution of hyperparameters by assuming that an optimization function obeys the Gaussian distribution [40]–[42]. Gaussian processes are used to fit data and update the posterior distribution of an objective function because they are highly flexible and simple to handle [40]. The problem solved with Bayesian optimization can be written in the form of Expression (2.28) [41], [42]:

$$\max_{\mathbf{x}\in\mathbf{A}\subset\mathbb{R}^d} f(\mathbf{x}), \tag{2.28}$$

where f is an objective function evaluated at $\mathbf{x}$ and d is the dimension of $\mathbf{x}$ which is typically less than 20 for most successful applications of Bayesian optimization [41]. $\mathbf{A}$ is a hyper-rectangle or a d-dimensional simplex.

When using Bayesian optimization, the following assumptions are made [42]:

- Evaluating $f(\mathbf{x}_i)$ is expensive and derivatives and convexity properties are unknown.
- Observations of f at sampled points can be obtained.
- f is Lipschitz-continuous. That is, for all $\mathbf{x_1}, \mathbf{x_2} \in \mathbf{A}$,

$$||f(\mathbf{x_1}) - f(\mathbf{x_2})|| \leq C||\mathbf{x_1} - \mathbf{x_2}||, \tag{2.29}$$

  where C is a constant which is most of the times unknown.
- There is a global maximum $\mathbf{x}^{(*)}$ such that for all $\mathbf{x}$, $f(\mathbf{x}^{(*)}) \geq f(\mathbf{x})$.

Bayesian optimization is made up of two main components: The Bayesian statistical model and the acquisition function. The Bayesian statistical model models an objective function in the following manner:

- Let $\mathbf{x}_i$ be the i^{th} sample and $f(\mathbf{x}_i)$ be the value of an objective function at $\mathbf{x}_i$.
- When the i^{th} observation $J_i = \{\mathbf{x}_i, f(\mathbf{x}_i)\}$ is acquired, a posterior distribution $P(f|J_i)$ is obtained by combining a prior distribution $P(f)$ with the likelihood function $P(J_i|f)$ using the Bayes' theorem as shown by Equation (2.30):

$$P(f|J_i) = \frac{P(J_i|f)P(f)}{P(J_i)}. \tag{2.30}$$

The acquisition function decides the next sample and guides the search towards the optimum [41], [42]. The objective function is evaluated at the next point $\mathbf{x} > \mathbf{x}_i$ where the acquisition function is maximized ($\arg\max_{\mathbf{x}_i} a(\mathbf{x}|J), a(\cdot)$ is an acquisition function) [42].

Bayesian optimization is very efficient in terms of the number of function evaluations performed because it incorporates prior information about the objective function to direct sampling and it performs the trade-off between exploring and exploiting the search space [42].

2.3 Dimensionality Reduction

In this research, frequency-domain and time-domain features were extracted from univariate time-series data. Each observation was represented by all features. Features' representation of data helps in simplifying classification, however, if the features are too many, the representation makes it harder to interpret the data and train the classifier, hence the need for dimensionality reduction.

Dimensionality reduction assumes that high-dimensional data is an indirect representation of low-dimensional manifold [43]. To find the low-dimensional manifold, many algorithms have been developed. Contributions to the development of these algorithms come from different fields with different approaches [43]. In this research, dimensionality reduction was achieved by projecting extracted features dimension to a reduced dimension of principal components using PCA. It was also done by selecting a subset of features using mRMR to analyse features' contribution to the classification task. PCA and mRMR details are given in subsections 2.3.1 and 2.3.2 respectively.

2.3.1 Principal Component Analysis

PCA is a non-parametric method that is used to extract relevant information from a complex dataset, thereby reducing its dimension to reveal hidden, simplified structures that underlie it [44]. It reduces the large dataset dimensionality in an interpretable manner while preserving much of the statistical information [45]. Statistical information preservation is achieved by finding new variables (called principal components (PCs)) that are linear functions of variables (or features) in the original dataset, uncorrelated to one another and maximizing variance [45]–[48].

Foundations of PCA can be traced back to the early 1900s when Karl Pearson published a paper that explored ideas to best fit a series of points to an affine space using the sum of squared orthogonal distances from each point to the space [49]. It is now used for different statistical applications in many fields including Biology and Finance [49].

The main goals achieved with PCA are as follows [46], [50]:

- Extraction of most important information from data/feature table, thereby compressing and simplifying dataset description, and
- Analysis of observations and variables' structure.

For dimensionality reduction purposes, the first $r \leq m$ PCs that retain acceptable variance can accurately represent feature matrix $\mathbf{X}$ in a reduced r-dimensional subspace.

The details of PCA are as follows [44], [46], [49], [50]:

Let $\mathbf{X}$ be a matrix of size $m \times n$ with m features and n observations. Let $\mathbf{Y}$ be an $m \times n$ representation of original features $\mathbf{X}$, mapped to $\mathbf{X}$ by a transformation $\mathbf{Z}$ such that $\mathbf{Y} = \mathbf{ZX}$.

Also let:

- $\mathbf{X}^{(i)}$ represent the i_{th} column of $\mathbf{X}$,
- $\mathbf{Y}^{(i)}$ represent the i_{th} column of $\mathbf{Y}$, and
- $\mathbf{Z}_{(i)}$ represent the i_{th} row of $\mathbf{Z}$.

$\mathbf{Y} = \mathbf{ZX}$ can be re-written as

$$\begin{bmatrix} \mathbf{Z}_{(1)}\dot{\mathbf{X}^{(1)}} & . & . & . & \mathbf{Z}_{(1)}\dot{\mathbf{X}^{(n)}} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ \mathbf{Z}_{(m)}\dot{\mathbf{X}^{(1)}} & . & . & . & \mathbf{Z}_{(m)}\dot{\mathbf{X}^{(n)}} \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_{(1)} \\ . \\ . \\ . \\ \mathbf{Z}_{(m)} \end{bmatrix} \begin{bmatrix} \mathbf{X}^{(1)} & ... & \mathbf{X}^{(n)} \end{bmatrix}. \tag{2.31}$$

From Equation (2.31), $\mathbf{Y}^{(i)} = \begin{bmatrix} \mathbf{Z}_{(1)}\dot{\mathbf{X}}^{(i)} \\ . \\ . \\ . \\ \mathbf{Z}_{(m)}\dot{\mathbf{X}}^{(i)} \end{bmatrix}$.

The covariance matrix $\mathbf{C_X}$ of $\mathbf{X}$ is given by

$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^\intercal$. It has the following properties:

- It is an $m \times m$ matrix that is square symmetric.
- Its diagonal terms are the variance of particular features.
- The off-diagonal terms are the covariance between features.

To reduce dimension, PCA aims to minimize the off-diagonal terms and maximize diagonal terms of the covariance matrix $\mathbf{C_Y}$. The optimized $\mathbf{C_Y}$ should have the following properties:

- All off-diagonal terms should be zeros ($\mathbf{C_Y}$ should be a diagonal matrix).
- Its dimensions should follow one another in order of descending variance.

To make $\mathbf{C_Y}$ a diagonal matrix, PCA assumes that $\mathbf{Z}$ is an orthonormal matrix. $\mathbf{C_Y}$ is diagonalized using the following algorithm:

- A normalized direction in the m-dimensional space along which the variance in $\mathbf{X}$ is maximized is selected and saved as a row vector $\mathbf{Z}_{(1)}$.
- Another normalized direction orthogonal to the first selected direction along which variance is maximized is selected and saved as $\mathbf{Z}_{(2)}$.
- The procedure is selected among all directions orthogonal to all previously selected directions, each time finding a direction along which variance is maximized, and saving it as $\mathbf{Z}_{(i)}$ until m vectors have been selected.

The resultant matrix $\mathbf{Z}$ is a matrix of PCs. The algorithm for diagonalizing $\mathbf{C_Y}$ is done through Eigenvector decomposition or singular value decomposition (SVD) [51].

When PCA is applied to a matrix $\mathbf{X}$ of size $m \times n$ with m features and n onservations, m PCs $\{c\}_{i=1}^{m}$ are obtained, which are ordered in descending order with respect to their variances[50]. A PC at position p is given by

$$c_p = \arg\max_{||c||=1} ||(\mathbf{X}^\intercal - \sum_{i=1}^{p-1} \mathbf{X}^\intercal c_i c_i^\intercal)c_i||, \tag{2.32}$$

and its variance is obtained by evaluating $||\mathbf{X}^\intercal c_p||^2$.

Eigenvalue Decomposition

Eigenvalue decomposition deals with breaking down of a square matrix $\mathbf{A}$ into its eigenvalues and eigen vectors [52], [53]. Eigenvalue λ and eigen vector $\mathbf{v}$ of $\mathbf{A}$ satisfy Equation (2.33) given by:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}, \tag{2.33}$$

where λ is a scalar and $\mathbf{v} \in \mathbb{R}$ is a non-zero vector. Eigenvalues/vectors are also known as character roots/vectors and latent roots/vectors respectively [52], [53].

Let $\mathbf{A}$ be an $n \times n$ matrix given by $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ a_{n1} & a_{n2} & a_{n3} & \cdot & a_{nn} \end{bmatrix}$.

Equation (2.33) can be expressed in the matrix form given by Equation (2.34):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ a_{n1} & a_{n2} & a_{n3} & \cdot & a_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix} = \lambda \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix}, \tag{2.34}$$

which is equivalent to:

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} & \cdots & a_{2n} \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ a_{n1} & a_{n2} & a_{n3} & \cdot & a_{nn} - \lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{bmatrix}. \tag{2.35}$$

Equation (2.35) can be re-written as $(A - \lambda \mathbf{I})\mathbf{v} = 0$.

Eigenvalue decomposition utilizes the following Linear Algebra theorems [44]:

- If $\mathbf{A}$ is orthogonally diagonalizable, then $\mathbf{A}$ is a symmetric matrix that can be expressed by Equation (2.36):

$$\mathbf{A} = \mathbf{BDB}^{\intercal}, \tag{2.36}$$

  where $\mathbf{D}$ is a diagonal matrix and $\mathbf{B}$ is a matrix that diagonalizes $\mathbf{A}$.

- A symmetric matrix $\mathbf{A}$ is diagonalized by a matrix of its othonormal eigenvectors. i.e., Let $\mathbf{A}$ in Equation (2.36) be an $n \times n$ matrix with eigen vectors $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, ..., \mathbf{v_n}$. Then $\mathbf{B} = [\mathbf{v_1 v_2 v_3 ... v_n}]$.

- Let $\mathbf{A}$ be an $n \times n$ orthogonal matrix. Then $\mathbf{A}^{-1} = \mathbf{A}^{\intercal}$.

When finding the principal components using eigenvalue decomposition, again let $\mathbf{X}$ be an input matrix of size $m \times n$ with m features and n observations. The

goal is to find an othonormal matrix $\mathbf{Z}$ in $\mathbf{Y} = \mathbf{ZX}$ such that $\mathbf{C_Y} = \frac{1}{n}\mathbf{YY}^\intercal$ is a diagonal matrix. The rows of $\mathbf{Z}$ are the principal components of $\mathbf{X}$ [44].

$\mathbf{C_Y}$ can be written interms of unknown matrix $\mathbf{Z}$ as follows:

$$\begin{aligned}\mathbf{C_Y} &= \frac{1}{n}\mathbf{YY}^\intercal \\ &= \frac{1}{n}(\mathbf{ZX})(\mathbf{ZX})^\intercal \\ &= \frac{1}{n}\mathbf{ZXX}^\intercal\mathbf{Z}^\intercal \\ &= \mathbf{Z}(\frac{1}{n}\mathbf{XX}^\intercal)\mathbf{Z}^\intercal \\ &= \mathbf{ZC_X Z}^\intercal.\end{aligned} \tag{2.37}$$

Now the matrix $\mathbf{Z}$ should be selected in such a way that each of its rows $\mathbf{Z}_{(i)}$ is an eigen vector of $\mathbf{C_X}$.

$\mathbf{C_Y}$ can then be evaluated by Equation (2.38).

$$\mathbf{C_Y} = \mathbf{ZC_X Z}^\intercal = \mathbf{Z}(\mathbf{ZDZ}^\intercal)\mathbf{Z}_\intercal = (\mathbf{ZZ}^\intercal)\mathbf{D}(\mathbf{ZZ}^\intercal) = (\mathbf{ZZ}^{-1})\mathbf{D}(\mathbf{ZZ}^{-1}) = \mathbf{D}. \tag{2.38}$$

All in all, the principal components of $\mathbf{X}$ are the eigen vectors of $\mathbf{C_X}$, and the variance of $\mathbf{X}$ along $\mathbf{Z}_{(i)}$ is the i_{th} diagonal value of $\mathbf{C_Y}$.

Singular Value Decomposition

SVD is a method of data dimensionality reduction by approximating original high-dimensional data points in a reduced dimensional space [52]. It can be used to identify and order the dimensions along which data points have the most variance to the least variance. SVD is founded on the following theorem from Linear Algebra:

- Let $\mathbf{A}_{mn}$ be a rectangular matrix of size $m \times n$, then it can be represented

by Equation (2.39).

$$\mathbf{A}_{mn} = \mathbf{U}_{mm}\mathbf{S}_{mn}\mathbf{V}^{\intercal}_{mn}, \tag{2.39}$$

such that $\mathbf{U}_{mm}$ is an $m \times m$ orthogonal matrix whose columns are eigen vectors of $\mathbf{A}\mathbf{A}^{\intercal}$, $\mathbf{V}_{nn}$ is an $n \times n$ orthogonal matrix whose columns are eigen vectors of $\mathbf{A}^{\intercal}\mathbf{A}$, and $\mathbf{S}_{mn}$ is an $m \times n$ matrix which contains the ordered square roots of eigen vectors of $\mathbf{U}$ or $\mathbf{V}$ (Matrices $\mathbf{U}$ and $\mathbf{V}$ have the same eigen values) in descending order [51], [52]. The columns of $\mathbf{U}$ are called *left singular vectors of* $\mathbf{A}$ and the columns of $\mathbf{V}$ are called *right singular vectors of* $\mathbf{A}$ [51].

Eigenvalue decomposition is defined for square matrices only; SVD generalizes eigenvalue decomposition to analyze all rectangular matrices [51]. A *positive semi-definite* matrix is a resultant of a matrix by its transpose (i.e., $\mathbf{A}\mathbf{A}^{\intercal}$ is a positive semi-definite matrix) [51]. Given that a matrix $\mathbf{X}$ is positive semi-definite, it can be eigenvalue decomposed and expressed as in Equation (2.40):

$$\mathbf{X} = \mathbf{R}\mathbf{T}\mathbf{R}^{\intercal}, \tag{2.40}$$

where $\mathbf{R}$ is an orthogonal matrix and $\mathbf{T}$ is a diagonal matrix containing eigenvalues of $\mathbf{X}$. SVD can be seen in terms of eigenvalue decomposition of the semi-definate matrices $\mathbf{A}\mathbf{A}^{\intercal}$ and $\mathbf{A}^{\intercal}\mathbf{A}$ as shown by Equations (2.41) and (2.42):

$$\mathbf{A}\mathbf{A}^{\intercal} = \mathbf{U}\mathbf{S}\mathbf{V}^{\intercal}\mathbf{V}\mathbf{S}\mathbf{U}^{\intercal} = \mathbf{U}\mathbf{S}^2\mathbf{U}^{\intercal} = \mathbf{U}\mathbf{T}\mathbf{U}^{\intercal}, \tag{2.41}$$

$$\mathbf{A}^{\intercal}\mathbf{A} = \mathbf{V}\mathbf{S}\mathbf{U}^{\intercal}\mathbf{U}\mathbf{S}\mathbf{V}^{\intercal} = \mathbf{V}\mathbf{S}^2\mathbf{V}^{\intercal} = \mathbf{V}\mathbf{T}\mathbf{V}^{\intercal}. \tag{2.42}$$

From Equations (2.41) and (2.42), it can be seen that $\mathbf{T} = \mathbf{S}^2$. $\mathbf{U}$ is a matrix composed of eigen vectors of $\mathbf{A}\mathbf{A}^{\intercal}$ and $\mathbf{V}$ is a matrix composed of eigen vectors of $\mathbf{A}^{\intercal}\mathbf{A}$ [51].

2.3.2 Feature Selection - Minimum Redundancy Maximum Relevance

Feature selection is used to identify the most prominent features in a dataset to ease the training task and minimize classification error [54].

The minimum redundancy maximum relevance (mRMR) scheme is a combination of *maximal relevance* feature selection scheme and *minimum redundancy* condition with the following details.

Maximum Relevance

Given a target class $\mathbf{o}$, this scheme selects features with the highest relevance to $\mathbf{o}$ [54]. Relevance is usually characterized by correlation or mutual information. With mutual information as a measure of relevance, the features identified as the most prominent ones should have the largest mutual information $\mathbf{MI}(\mathbf{X}_{(i)}, \mathbf{o})$ with the target class $\mathbf{o}$, thereby showing the largest dependency on $\mathbf{o}$. Mutual information between variables $\mathbf{A}$ and $\mathbf{B}$ is given by

$$\mathbf{MI}(\mathbf{A}, \mathbf{B}) = \sum_{a \in \mathbf{A}} \sum_{b \in \mathbf{B}} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}. \tag{2.43}$$

For all features $\{\mathbf{X}_{(i)}\}$, maximum relevance R_l is implemented using mean value of their mutual information with a target class $\mathbf{o}$. i.e.,

$$R_l = \frac{1}{|X|} \sum \mathbf{MI}(\mathbf{o}, \mathbf{X}_{(i)}). \tag{2.44}$$

With maximum relevance alone, the combination of best features does not necessarily lead to good classification performance. They can have high dependency among themselves (high redundancy) [54]–[56]. To minimize redundancy, minimum redundancy is combined with maximum relevance.

Minimum Redundancy

Minimum redundancy R_d is used to select features that are mutually maximally dissimilar [54], [57]. It is given by:

$$R_d = \frac{1}{|X|^2} \sum \mathbf{MI}(\mathbf{X}_{(i)}, \mathbf{X}_{(j)}), \tag{2.45}$$

where $\mathbf{X}_{(i)}, \mathbf{X}_{(j)} \in \mathbf{X}$.

mRMR feature selection goal is achieved by optimizing relevance and redundancy given by $\max(R_l - R_d)$ [54]–[58].

2.4 Conclusion

Techniques employed in this work to detect electricity theft were presented in this chapter. DNNs' training options including activation functions were discussed. First-order and derivative-free optimization techniques and their applications were presented, with emphasis on Adam and Bayesian optimization techniques respectively. Dimensionality reduction algorithms were also presented, with special emphasis on PCA and mRMR. The following chapter will go into the details of how the DNN and dimensionality reduction techniques presented here were used to solve detect electricity theft.

Chapter 3

DNN-based Electricity Theft Detection Scheme using Time-Frequency Domains Features and Dimensionality Reduction Techniques

In this chapter, a DNN-based electricity theft detection scheme is presented. It is based on the following publication: "*Electricity Theft Detection in Smart Grids based on Deep Neural Network*", which is under IEEE Access's second round of review.

The scheme utilizes time-domain and frequency-domain features in a fully connected feed-forward DNN classification-based approach to differentiate between faithful and unfaithful customers classes.

The author is responsible for the conceptualization of the work, experimental data collection, analysis and interpretation, as well as project execution and results collection. The work was done with assistance from the co-authors: Prof. Ling

Cheng and Mr. Shamin Achari.

3.1 Chapter Summary

Electricity theft is a global problem that negatively affects both utility companies and electricity users. It destabilizes the economic development of utility companies, causes electric hazards and impacts the high cost of energy for users. The development of smart grids plays an important role in electricity theft detection since they generate massive data that includes customer consumption data which, through machine learning and deep learning techniques, can be utilized to detect electricity theft. This chapter introduces the theft detection method which uses comprehensive features in time and frequency domains in a deep neural network-based classification approach. We address dataset weaknesses such as missing data and class imbalance problems through data interpolation and synthetic data generation processes. We analyse and compare the contribution of features from both time and frequency domains, run experiments in combined and reduced feature space using principal component analysis and finally incorporate minimum redundancy maximum relevance scheme for validating the most important features. We improve the electricity theft detection performance by optimizing hyperparameters using a Bayesian optimizer and employing an adaptive moment estimation optimizer to carry out experiments using different values of key parameters to determine the optimal settings that achieve the best accuracy. Lastly, we show the competitiveness of our method in comparison with other methods evaluated on the same dataset. On validation, we obtained 97% area under the curve (AUC), which is 1% higher than the best AUC in existing works, and 91.8% accuracy, which is the second-best on the benchmark.

The remainder of this chapter is organized as follows. Section 3.2 covers the related work done in literature to tackle the electricity theft problem. Section 3.3 covers step-by-step method taken in this work; which includes dataset analysis and work done to improve its quality, and customers' load profile analysis which led to features extraction and classification. In Section 3.4, the results are presented and discussed. The chapter conclusion follows in Section 3.5.

3.2 Related Work

Research on electricity theft detection in smart grids has attracted many researchers to devise methods that mitigate electricity theft. Methods used in the literature can be broadly categorized into the following three categories: hardware-based, combined hardware and data-based detection methods and data-driven methods.

Hardware-based methods [59]–[65] generally require hardware devices such as specialized microcontrollers, sensors and circuits to be installed on power distribution lines. These methods are generally designed to detect electricity theft done by physically tampering with distribution components such as distribution lines and electricity meters. They can not detect cyber attacks. An electricity cyber attack is a form of electricity theft whereby energy consumption data is modified by hacking the electricity meters [12].

For instance, in [59], an electricity meter was re-designed. It used components that include: a Global System for Mobile Communications (GSM) module, a microcontroller, and an Electrically Erasable Programmable Read-Only Memory (EEPROM). A simulation was done and the meter was able to send a Short Message Service (SMS) whenever an illegal load was connected by bypassing the meter. Limited to detecting electricity theft done by physically tampering with distribution components such as distribution lines and electricity meters. Authors in [62] used the GSM module, ARM-cortex M3 processor and other hardware components to solve the electricity theft problem done in the following four ways: bypassing the phase line, bypassing the meter, disconnecting the neutral line, and tampering with the meter to make unauthorized modifications. A prototype was built to test all four possibilities. The GSM module was able to notify with SMS for each theft case.

Authors in [63] designed ADE7953 chip-based smart meter which is sensitive to current and voltage tempering, and mechanical tempering. ADE7953 was used to detect overvoltage, dropping voltage, overcurrent, the absence of load and other irregularities in voltage and current. It sent an interrupt signal to the Microcontroller Unit (MCU) which reported tampering status. Mechanical

tampering was overcome by connecting a tampering switch to MCU's IO ports so that it can send alarm signals to MCU once tampered with. The design was tested with tampering cases such as connecting the neutral and the phase lines, connecting the meter input and output in reverse mode, and bypassing the phase line to load. The probability of detection failure was 2.13%.

Authors in [61] used a step-down transformer, voltage divider circuit, microchip and other hardware components to design a circuitry to detect electricity theft by comparing forward current on the main phase line with reverse current on the neutral line. The circuitry was installed before the meter.The design was tested on Proteus software and on actual hardware. When the meter was bypassed, the problem was detected and an alarm sounded. In [60], a circuit to detect electricity theft done by bypassing the meter was designed. The transformers, rectifiers, microcontroller, GSM module and other hardware components were used. The GSM controller notified the operator with SMS when the meter was bypassed.

Authors in [64] proposed putting the Radio Frequency Identification (RFID) tags on ammeters and capturing unique data about each ammeter. Ammeters were to be tracked and managed in real-time. Electricity theft was to be inspected on site. Damaged, removed or a tag with a different information from the original one means high possibility that an electricity theft happened. Evaluation based on analysis on cost of deployment. With a case study made on utility company in China, Return on Investment (ROI) was found to be greater than one. In [65], An Arduino-based real-time electricity theft detector was designed. The following hardware was used: Arduino Uno, GSM module, current sensors and LCD. The Arduino Uno obtained measurements from current sensors which were located one on the secondary side of the transformer and the other on the electric service cap. If the difference between current sensors' measurements exceeded a set threshold, the message would be sent to the operator via a GSM module. The simulation was done using Proteus 8 software and the prototype was built on hardware, which was able to report theft cases when tested.

Apart from their inability to detect cyber attacks, these methods are also expensive due to their need for special hardware deployment and maintenance. Combined hardware and data-based electricity theft detection methods [66]–[68]

employ the use of hardware, machine learning and/or deep learning techniques to tackle the electricity theft problem. Due to hardware requirements, these methods also pose the challenge of being expensive to deploy and maintain.

In [66], a method to measure the total consumption of a neighbourhood and compare the results with the usage reported by the smart meters in that neighbourhood was proposed. A significant difference between smart meters' and transformers' measurements would mean the presence of unfaithful customers in the neighbourhood. To locate the unfaithful customers in the neighbourhood, the authors proposed using a Support Vector Machine (SVM) classifier. The classifier was tested on a dataset of 5000 (all faithful) customers. A maximum detection rate of 94% and a minimum false positive rate of 11% were achieved.

Authors in [68] developed a predictive model to calculate TLs. To get NTL, TLs would be subtracted from total distribution network losses. Based on an assumption that distribution transformers and smart meters share data to the utility after every 30 minutes, a smart meter simulator was used to generate data for 30 users in 30 minutes intervals for 6 days. On the simulator, unfaithful users stole electricity by bypassing the meter. Stolen electricity was varied between 1% and 10% of the total consumption. For stolen electricity value over 4%, the detection rate was 100%, which diminished as stolen electricity percentage was decreased.

In [67], a method which would use an observer meter that would be installed on a pole away from households and record the total amount of electricity supplied to n households where it is suspected that one or more meters have been tampered with was proposed. The observer meter would have camera surveillance to protect it from being tampered with. A mathematical algorithm that utilizes data from an observer meter and smart meters to detect a smart meter tempered with was developed. A mathematical algorithm was tested with a real-world consumption dataset by increasing the consumption of some meters which were picked randomly. The algorithm was able to detect the meters with altered consumption.

Due to high-cost demand in the above categories, many researchers work on data-driven methods to overcome the electricity theft problem. For instance, the au-

thors in [10] designed an electricity theft detection system by employing three algorithms in the pipeline: Synthetic Minority Over-sampling Technique (SMOTE), Kernel function and Principal Component Analysis (KPCA) and SVM. They used SMOTE to generate synthetic data for balancing an unbalanced dataset, KPCA to extract features and SVM for classification. They obtained maximum overall classifier quality characterized by Area Under the Curve (AUC) of 89% on validation.

Authors in [11] used wide and deep Convolutional Neural Networks (CNN) model to detect electricity theft. Based on that normal electricity consumption is periodical while stolen electricity consumption data is not periodical, wide was to learn multiple co-occurrences of features for 1-D series data, while deep CNN was used to capture periodicity with data aligned in a 2-D manner by weeks. They varied training and validation data ratios, to obtain maximum AUC value of 79%. By utilizing the same dataset used in [10] and [11], the method we present in this work achieves AUC results beyond 90% on both validation and testing.

In [50], PCA was used to transform original high-dimensional consumption data by extracting Principal Components (PCs) which retained the desired variance. An anomaly score parameter that was defined between set minimum and maximum thresholds was introduced. For each test sample, the anomaly score parameter was calculated. If the result was not between the set thresholds, the sample would then be treated as malicious. The true positive rate (TPR) was used to evaluate the method, which hit the best-recorded value of 90.9%. Authors in [69] used One-Class SVM (O-SVM), Cost-Sensitive SVM (CS-SVM), Optimum Path Forest (OPF) and C4.5 tree. From customer consumption data, different features were selected, and the performance of each classifier was analyzed independently on a different set of features, followed by combining all classifiers for the best results. Best results were achieved when all classifiers were combined, with 86.2% accuracy.

Authors in [70] employed a combination of CNN and Long Short-Term Memory (LSTM) recurrent neural network deep learning techniques. Seven hidden layers were used, of which four of them were used by CNN and three were utilized by LSTM. This method relied on CNN's automatic feature extraction ability on a given dataset. Features were extracted from 1-D time-series data. On model

validation, the maximum accuracy achieved was 89%. The authors in [71] used a combination of Local Outlier Factor (LOF) and k-means clustering to detect electricity theft. They used k-means clustering to analyze the load profiles of customers, and LOF to calculate the anomaly degrees of customers whose load profiles were from their cluster centres. On the evaluation of the method, they attained an AUC of 81.5%. Our model achieves a maximum value of 91.8% accuracy and 97% on validation.

In [72], two electricity theft models were developed. The first model is based on Light Gradient Boosting (LGB) classifier. A combination of SMOTE and Edited Nearest Neighbour (ENN) was used to balance the dataset. Feature extraction was done using AlexNet, followed by classification with LGB. This proposed model was named as SMOTEENN-AlexNet-LGB (SALM). The second model is based on the Adaptive Boosting classifier. Conditional Wasserstein Generative Adversarial Network with gradient penalty (CWGAN-GP) was used to generate synthetic data that resembled the minority class data to balance data of the unbalanced classes. Feature extraction was performed using GoogleNet, then classification by AdaBoost followed. The proposed model was named as GAN-NETBoost. The models were evaluated with SGCC data used in this work. SALM and GAN-NetBoost attained an accuracy of 90% and 95%, and AUC of 90.6% and 96% respectively on validation.

Although these models were able to achieve impressive results, their consideration of time-domain features alone limited their performance. Our solution shows that adding frequency-domain features on time-domain features improves classification performance.

3.3 DNN-Based Electricity Theft Detection Method

The electricity theft detection method outlined in this section consists of the following three steps: Data Analysis and Pre-processing, Feature Extraction, and Classification. Figure 3.1 shows the workflow diagram.

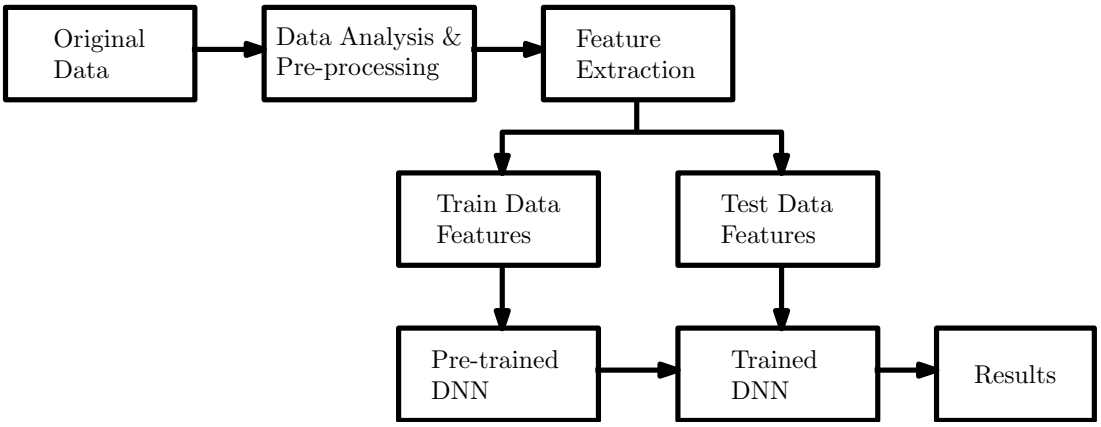


Figure 3.1. Electricity theft detection workflow diagram

3.3.1 Data Analysis and Pre-processing

In this sub-section, we present the dataset used and its quality improvement by identifying and removing records that had no consumption data. In this work, an observation refers to a single instance/record in the dataset, for the duration of measured consumption. i.e., given a dataset $\mathbf{A}$ of size N, $\mathbf{a}_i \in \mathbf{A}$; where $\mathbf{a}_i$ is the i^{th} observation of $\mathbf{A}$ and $1 \leq i \leq N$.

We show customers' load profiles analysis. We further present data interpolation and synthetic data generation details that have been undertaken.

Dataset Analysis and Preparation

TABLE 3.1. Dataset summary table

Number of observation days: 1,034

Customer Class	**Number Of Observations**		
	From Original Dataset	*After Removing Empty Observations*	*After Synthetic Data Generation*
Faithful	38,757	36,679	36,679
Unfaithful	3,615	3,579	36,679
Total	**42,372**	**40,258**	**73,358**

As stated in Section 1, we used a realistic electricity consumption dataset released by SGCC, which is accessible at [17]. The dataset consists of daily electricity consumption data taken from January 2014 to October 2016, summarized in

Table 3.1. The sampling rate of the data is uniform for every customer, it is one measurement per day; which corresponds to the total power consumption for that day.

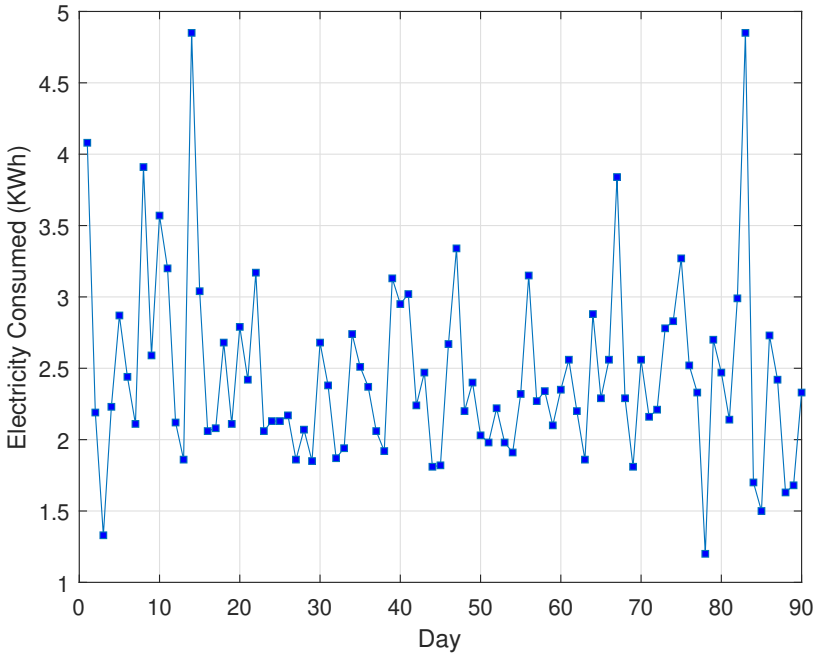
The used dataset consists of 42372 observations, of which 3615 observations are electricity consumption data of unfaithful customers and the remaining observations are electricity consumption data of faithful customers.

As with many datasets used in the literature, data comes with many errors caused by factors such as smart meters failures, data storage problems, data transmission issues and unscheduled systems maintenance [11]. Dataset used in this work is no exception. It consists of traces of non-numerical or null values. Using data analysis methods, we found approximately 5.45% of observations in this dataset to either have only null values, or zeros, or a combination of both, for the whole duration of 1034 days. These observations were regarded as *empty observations*. i.e., An observation $\mathbf{a}$ is regarded as an empty observation if $a_i = 0$ or $a_i \notin \mathbb{R}$ for all $a_i \in \mathbf{a}$.

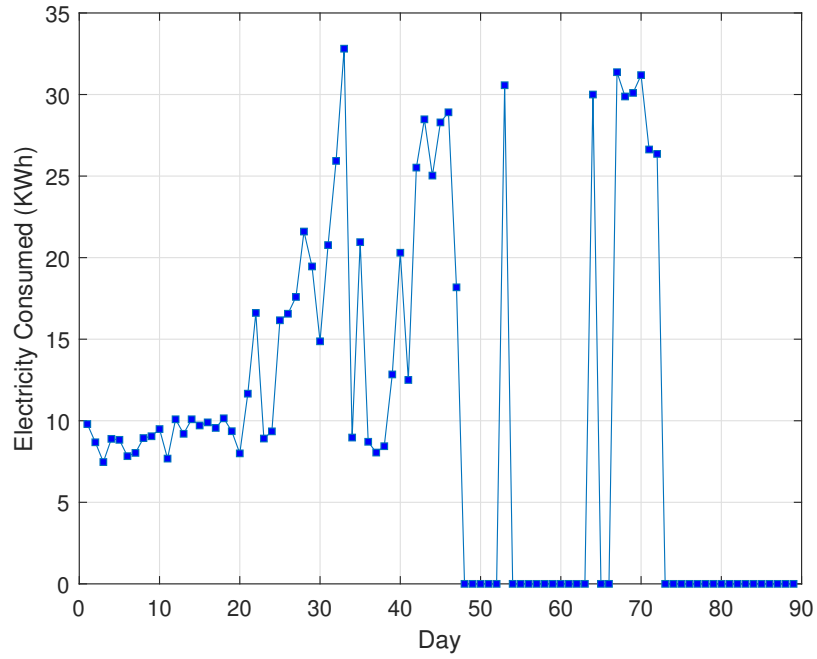
These observations do not have any differentiating characteristics between the classes since they do not have any consumed electricity record greater than 0 kWh. They could not be identified with any class as they were labelled with either of the classes, therefore they were discarded. The third column of Table 3.1 shows a summary of observations left after the removal of empty observations.

Figure 3.2 shows line plots of consumption data of a faithful customer and an unfaithful customer against the consumption days, for the duration of three months. Comparing the two graphs, we observed that the consumption behaviour of the honest customer is mostly uniform and has a predictable trend, while electricity thief's consumption behaviour takes different forms and is not predictable. We further carried out histogram analyses for both classes of customers, as shown in Figure 3.3.

From the histograms shown, we observe that for faithful customer's consumption data, statistical parameters mean, mode, and median are generally closer to the histogram centre as compared to unfaithful customer's consumption data. We did a similar analysis for many customers and found that an observation presented



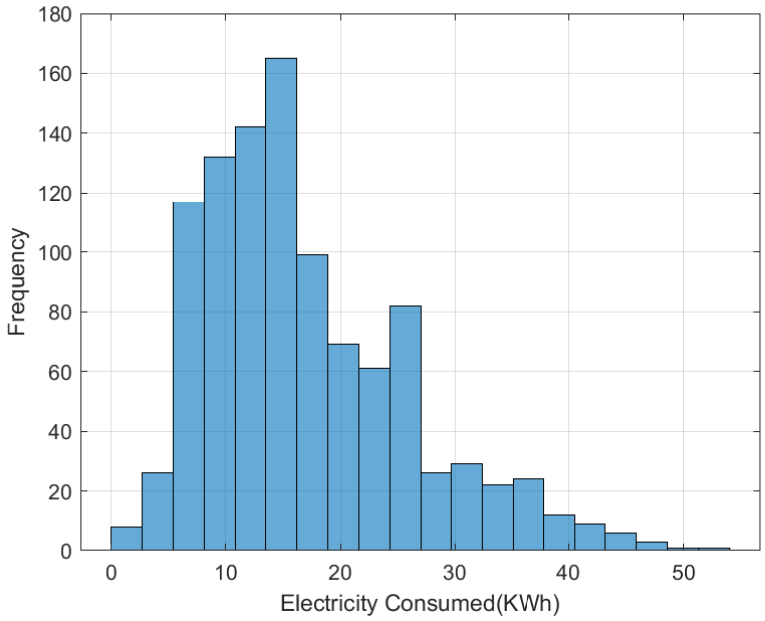
(a) Faithful customer's consumption plot



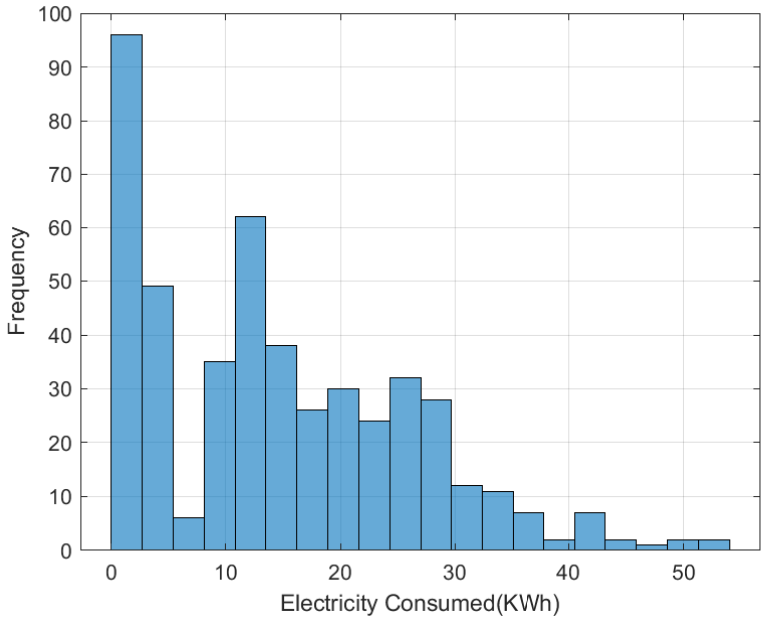
(b) Unfaithful customer's consumption plot

Figure 3.2. Faithful and unfaithful customers' consumption plots

here is true for most of the dataset. From these observations, we argue that by defining outliers as values beyond three Median Absolute Deviations (MAD), honest customers can be characterized as having fewer outliers percentage in a



(a) Faithful customer's consumption histogram



(b) Unfaithful customer's consumption histogram

Figure 3.3. Faithful and unfaithful customers' consumption histograms

given data, than unfaithful customers.

Data Interpolation

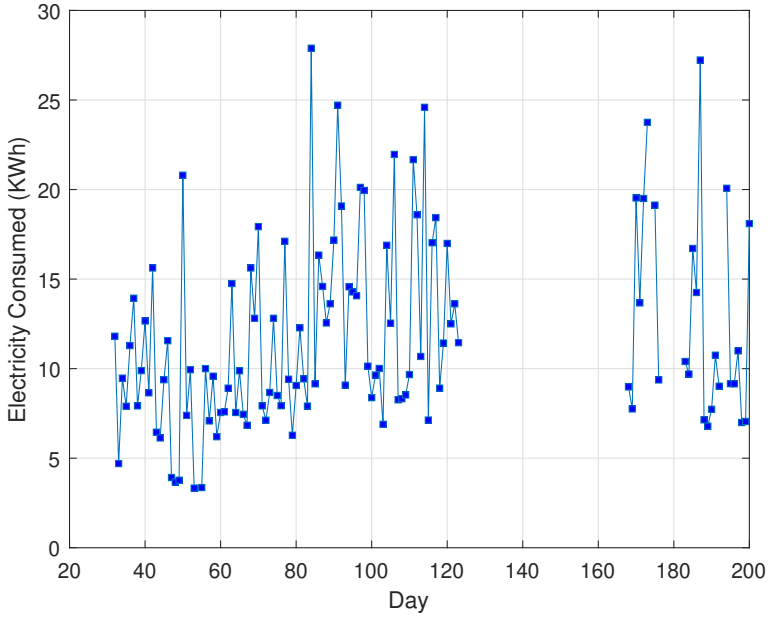
For all observations consisting of a combination of null or non-numerical values and real consumption values, data were interpolated. Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [73] was used to fill in missing data during data interpolation while preserving consumption patterns.

A cubic Hermite interpolating polynomial $H(x)$ is a shape-preserving interpolant which preserves data monotonicity on a sub-interval $x_i \leq x \leq x_{i+1}$ applied to. For the data consumption vector containing NaN values at the beginning, the raw data *mean* was evaluated excluding NaN values and then inserted as the first vector element. The rest of the elements were filled in using PCHIP. This helped to maintain consumption shape and avoided adding outliers to data.

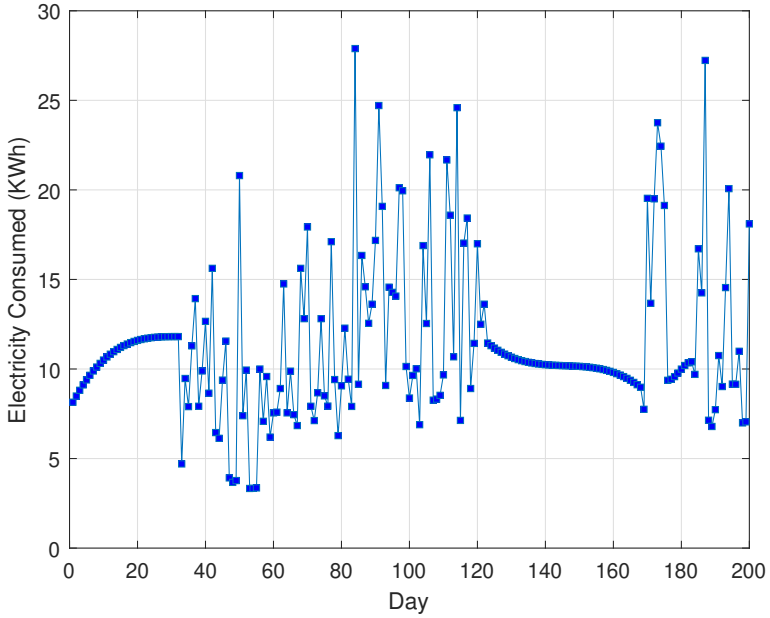
Figure 3.4 shows an example of one observation taken randomly before and after interpolation. A consumption duration of 200 days around days with missing consumption data is shown for clear presentation. Interpolated data points make a smooth curve that lies between the minimum and maximum near points with no overshooting, as can be seen from Figure 3.4b. In this manner, the consumption data is preserved from the addition of outliers and data points that can make interpolated data pattern to resemble unfaithful customer's consumption pattern such as the one shown in Figure 3.2b.

Synthetic Data Generation

After eliminating empty observations and interpolating data, we carried out the initial classification process. Experimenting with the dataset as is, we observed that the classifier satisfactorily classified faithful customers and performed badly on unfaithful customers due to a *class imbalance problem* [66],[70]. A class imbalance problem is a situation whereby the number of observations in one class is much greater than the number of observations in the other class. In a class imbalanced problem, classification models classify the majority class on a dataset successfully, while performing badly on the minority class [70]. Dataset used in this work has faithful customers number that is much greater than that of unfaithful customers.



(a) Consumption data before interpolation



(b) Consumption data after interpolation

Figure 3.4. Plots of consumption data before and after interpolation

We solved the class imbalance problem in the following manner:

1. Define q and r as the number of faithful and unfaithful customers respectively and evaluate the difference $p = q - r$.

2. From a set of faithful customers observations, randomly select p observations represented by $p \times 1034$ matrix $\mathbf{O}$ defined by Equation (3.1), such that

$$\mathbf{O} = \begin{bmatrix} o_{1,1} & o_{1,2} & \cdots & o_{1,1034} \\ o_{2,1} & o_{2,2} & \cdots & o_{2,1034} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ o_{p,1} & o_{p,2} & \cdots & o_{p,1034} \end{bmatrix}. \tag{3.1}$$

3. Inspired by [66] and dataset analysis observations in 3.3.1, we evaluated synthetic observations $\mathbf{O}^s$ by Hadamard product in Equation (3.2):

$$\mathbf{O}^s = \mathbf{C} \odot \mathbf{O} \tag{3.2}$$

where $\mathbf{C}$ is a matrix of randomly generated numbers of size $p \times 1034$ with elements between 0 and 1.

This helps to distort the pattern of consumption as observed through faithful customers' consumption data line plots shown in Section 3.3.1; hence the result better represents unfaithful customers' consumption data.

This approach of generating synthetic is cheap and fast to do as it uses the available data of faithful customers' class to generate data for the opposite class. It involves a single operation on the measured data, which is the multiplication of measured data by a matrix of randomly generated numbers. The resulting data was added to the original dataset, labelling it as belonging to unfaithful customers' consumption class. The fourth column in Table 3.1 shows a summary of observations after synthetic data generation.

3.3.2 Feature Extraction

Electricity consumption data used in this project is *univariate* time-series data. A univariate measurement is a single measurement frequently taken over time [74].

For solving classification problems, data can be represented by its features (properties), which can then be fed as input to the classifier, as is the case in [22],[27] and [75]. Data is classified based on the similarity between features [74] given a dataset of different samples. In this work, time-domain and frequency-domain features were extracted and used as input to a deep neural network for classification. Classification performance comparison between time-domain, frequency-domain and combined features from both domains was carried out.

TABLE 3.2. Time-domain and frequency-domain features table

Time-domain features	**Frequency-domain features**
Standards probability (stdsProb)	Harmonic frequency (hfInd)
Standards mean (stdsMean)	Harmonic frequency amplitude (hfAmp)
Standards standard deviation (stdsDev)	99% spectrum bandwidth (bww99)
Outliers probability (outsProb)	Lower bound frequency (flb)
Outliers mean (outsMean)	Upper bound frequency (fub)
Outliers standard deviation (outsDev)	99% bandwidth power (bwwpwr)
Data mean (dataMean)	50% bandwidth (bw50)
Data mode (dataMode)	Median frequency (fmed)
Data median (dataMedian)	Mean frequency (fmean)
Average of pchip interpolant curve fitted parameters (cfpMean)	

Time-domain Feature Extraction

As shown in Section 3.3.1, faithful and unfaithful customers' consumption data differs clearly by a pattern of consumption, as shown by line plots and histogram graphs. Based on this information, time-domain features stipulated in Table 3.2 can collectively be used to differentiate between the two classes of customers. Apart from an observation that consumption data of faithful customers roughly follow a predictable pattern, and unfaithful customers' consumption behaviour is not predictive, as shown in Figure 3.2, customers do not consume an equal amount of energy per given time.

Energy needs per customer may differ due to different reasons such as the number of appliances used, kind of appliances per household, household size, etc. To achieve higher accuracy in classifying features, all observations are made to fit within the same axes. This is achieved by normalizing data for each observation using the Min-Max method [76] given by Equation (3.3). The Min-max method

shrinks the data between 0 and 1 while keeping the original consumption pattern.

$$f(x_i) = \frac{x_i - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})} \tag{3.3}$$

Frequency-domain Feature Extraction

Fourier theorem states that a periodic signal $x(t)$ can be represented by a summation of complex sinusoidal signals with frequencies that are an integer multiple of fundamental frequency f_T [77]. Using the Fourier theorem, the consumption data graphs shown in Section 3.3.1 can be seen as a time series signal that can be transformed into the frequency-domain by using Fourier transform. Represented in frequency-domain, we extracted frequency-domain features from each observation. Since neural networks are sensitive to diverse input data, using Equation (3.3), features were normalized after being extracted so that they could be fed as input to the classifier. Table 3.2 shows features extracted from both domains.

3.3.3 Classification

We built two classifier architectures in this work. We built the first architecture using manual search for hyperparameters tuning, while the second architecture was the result of hyperparameters tuning based on a Bayesian optimization algorithm.

Network architecture based on manual search

A fully connected feed-forward DNN architecture shown in Figure 3.5 was used for the classification process.

In order to avoid network underfitting and overfitting [28], the following rule of thumb methods [28], [78] were considered in the design of hidden layers of a deep neural network classifier shown in Figure 3.5:

- Number of hidden neurons should be between the size of the input layer

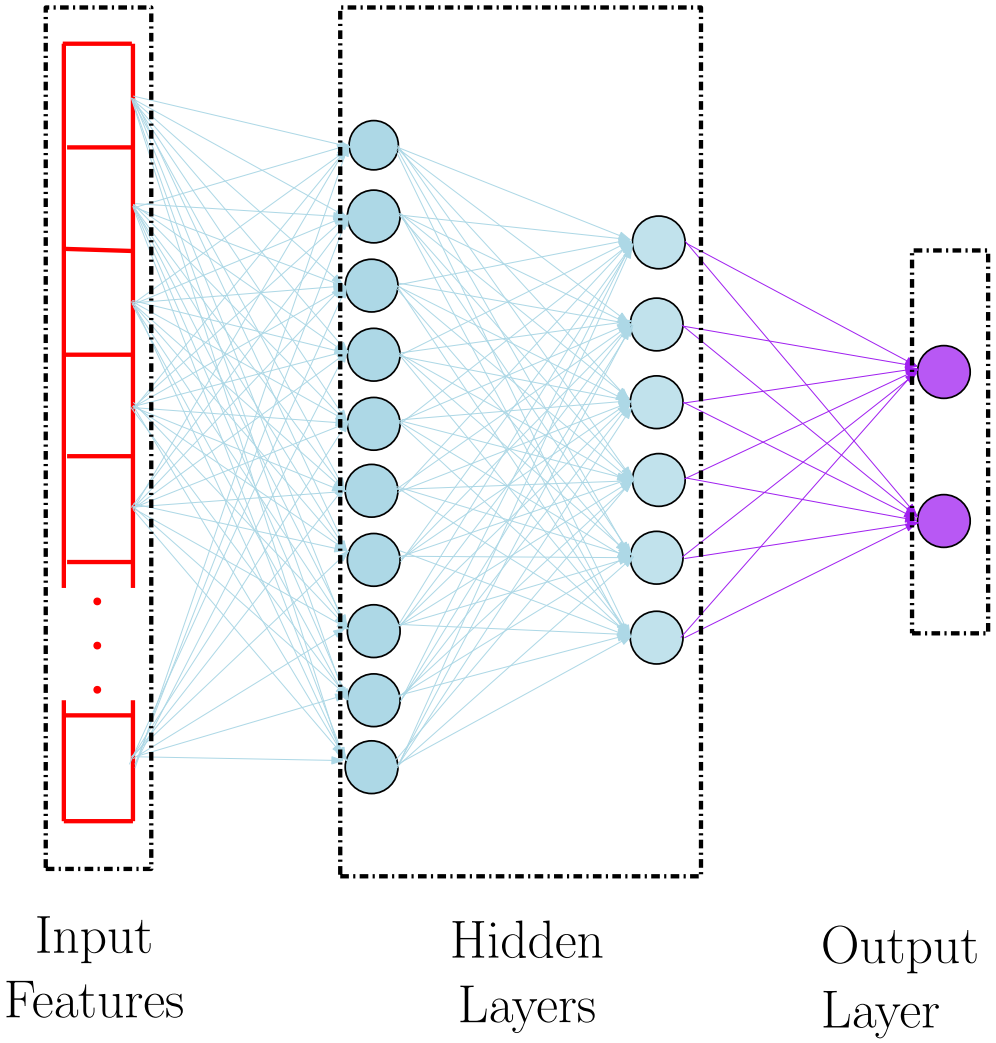


Figure 3.5. DNN classifier architecture

and size of the output layer,

- Number of hidden neurons should be approximated to the summation of $\frac{2}{3}$ size of input layer and size of the output layer.
- Number of hidden neurons should be less than twice the size of the input layer.

Rectified Linear unit (ReLU) activation function was used in the hidden neurons because of its better convergence property in comparison to other activation functions [21].

Training

The maximum number of training iterations was limited to 1000. The classification approach was split into four parts. In the first part, only time-domain features were used for classification. In the second part, only frequency-domain features were used. The third part comprised of combined features from both domains, while in the last part, classification was performed in reduced feature space by incorporating PCA.

Holdout validation scheme was used as follows: in all the procedures, as a rule of thumb, 80% of the whole data was used for training and validation, while 20% of the whole data was used for testing. Within training and validation data, 80% was used for training while 20% was used for validation. Similar results were obtained when using the k-fold cross-validation scheme with $k = 5$. More about using k-fold cross-validation scheme with $k = 5$ can be seen in [79] as an example.

During training, the aim of the model is to minimize the loss function so that the best results can be achieved. In this case, the DNN aimed to minimize the binary cross-entropy loss function [80]. Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm [81] was used to minimize the cross-entropy loss. The cross-entropy loss versus training iterations when the network in 3.5 was trained with all features from time-domain and frequency-domain is shown by Figure A.1 in Appendix A.

Performance Metrics

Based on true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) obtained from a confusion matrix [34], [82], we used the following performance metrics to evaluate the classifier's performance: Recall/True Positive rate (TPR) [83], [84], Precision/Positive Predictive Value (PPV) [83], [84], F1-Score [85], Matthews Correlation Coefficient (MCC) [70], Accuracy and Area Under the Curve of Receiver Operator Characteristic (AUC-ROC) curve [86].

A confusion matrix contains information about the actual and predicted classifications of a classifier [82]. In binary classification, an input is classified into only

one of two non-overlapping classes [87]. Figure B.1b in Appendix B shows test results in the form of a confusion matrix taken from classification done with all features from time-domain and frequency-domain.

We briefly introduce performance metrics used as follows.

Recall/True Positive Rate (TPR): is the measure of the fraction of positive examples that are correctly labelled. It is given by:

$$TPR = \frac{TP}{TP + FN}. \tag{3.4}$$

Precision/Positive Predictive Value (PPV): is the measure of the fraction of examples classified as positive that are truly positive. It is given by:

$$PPV = \frac{TP}{TP + FP}. \tag{3.5}$$

F1-Score: shows the balance between precision and recall. It is given by:

$$\textit{F1-Score} = 2 * \frac{TPR * PPV}{TPR + PPV}. \tag{3.6}$$

Accuracy: shows the fraction of predictions classified correctly by the model. It is given by:

$$\begin{aligned} \textit{Accuracy} &= \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \\ &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned}. \tag{3.7}$$

Matthews Correlation Coefficient (MCC): a single digit that measures a binary classifier's performance. Its value ranges from -1 to +1, with values closer to +1 signifying good performance, while values closer to -1 signify bad performance. MCC is given by:

$$\text{MCC} = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{3.8}$$

Area Under the Curve (AUC): measures the classifier's overall quality. Larger AUC values indicate better classifier performance.

Hyperparameters Optimization

To achieve the best classification performance at a reasonable amount of time, we used Bayesian optimization [40] to tune the following hyperparameters: number of hidden layers, size of each layer, regularization strength and activation function. To get the best combination of hyperparameters, 100 optimization steps were conducted. The resultant optimized network was trained and tested in a similar manner as the network in Figure 3.5.

Impact of Key Parameters Investigation

Using adaptive moment estimation (Adam) optimizer [37], an impact of the following three key parameters were investigated on the optimized network: initial learning rate, minibatch size and L2-regularization parameter. Data was divided into two parts: the training and validation data.

The volume of the training and validation/test data plays an important role in classification success. The higher the correlation between input features and the class label, the lesser the data needed for training [88]. However, given a dataset, the training data portion of less than 50% is not advised for as it will negatively affect the test results [88]. With this in mind, we, therefore, determined parameters' impact with different training data percentages.

We carried out the following procedure for 60%, 70% and 80% training data portions. For each parameter, its impact was investigated by determining training and validation accuracies with varied parameter values. Parameters were loga-

rithmically varied in 100 steps between the initial and final values. For each step, the number of training epochs was limited to 30. The other parameters were held at fixed values while adjusting a parameter under study. Table 3.3 shows investigated parameters' initial values, step values, final values as well as fixed values.

TABLE 3.3. Investigated parameters table

Parameter	**Initial Value**	**Log Step**	**Final Value**	**Fixed Value**
Initial learning rate	10^{-5}	$10^{0.03}$	10^{-2}	10^{-4}
Minibatch size	10^{1}	$10^{0.04}$	10^{5}	128
L2-regularization	10^{-8}	$10^{0.06}$	10^{-2}	10^{-5}

3.4 Results and Discussion

In this section, we show and discuss the experimental results. In Section 3.4.1, we present results obtained before synthetic data generation. In Section 3.4.2, we show a comparison between classification performance when using time-domain features, frequency-domain features and combined features from both domains as inputs to the classifier. We analyze PCA dimensionality reduction impact on experimental results in Section 3.4.3. We present Bayesian optimization results as well as best results attained with optimized classifier in Section 3.4.4, and we finally present an investigation of optimal parameter settings for best classification performance by varying different parameters using Adam optimizer in Section 3.4.5.

3.4.1 Validation Results Before Synthetic Data Generation

As stated in Section 3.3, when there is an imbalance in the number of observations between two classes, the classifier performs badly on the class with a relatively lower number of observations. The classifier shown in Figure 3.5 was trained with features extracted from an original dataset with no augmented synthetic data.

80% of the data was used for training while 20% was used for validation. The third column of Table 3.4 shows the validation results. For the faithful customers class, validation results are much better than the unfaithful class. This can be seen by a comparison between faithful and unfaithful customers recall, precision and F1-score shown.

Compared with validation results in combined domains before the incorporation of PCA, there was no significant change in the recall, precision and F1-score for faithful customers' class since the difference in corresponding values was within a 1% margin. However, for the unfaithful class, which was the minority class, validation results in terms of recall, precision and F1-score were not good at all before balancing the classes. A significant improvement was obtained after balancing the classes. This shows that the sensitivity of the classifier to the minority class was not as good as its sensitivity to the opposite class.

TABLE 3.4. Performance evaluation table

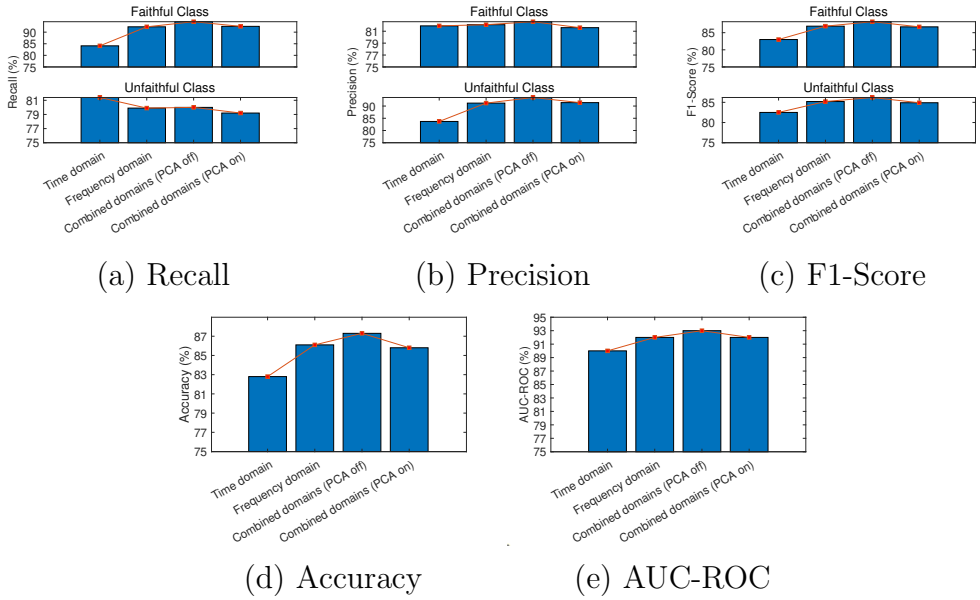
Parameter	**Class**	**Before synthetic data generation**	**After synthetic data generation**							
			Time-domain		**Frequency-domain**		**Combined Domains**			
							PCA Not Used		*PCA Used*	
		Val(%)	Val(%)	Test (%)	Val(%)	Test (%)	Val(%)	Test (%)	Val(%)	Test (%)
Recall	Faithful	94.6	85.8	84.1	92.8	92.3	94.2	94.5	93.0	92.5
	Unfaithful	4.3	89.2	81.4	90.4	79.9	90.0	80.0	89.0	79.2
Precision	Faithful	91.4	88.8	81.9	90.6	82.1	90.4	82.6	89.4	81.6
	Unfaithful	6.9	86.3	83.7	92.7	91.2	93.9	93.5	92.7	91.4
F1-Score	Faithful	93.0	87.3	83.0	91.7	86.9	92.3	88.2	91.2	86.7
	Unfaithful	5.3	87.7	82.5	91.5	85.2	91.9	86.2	90.8	84.9
Accuracy		86.9	87.5	82.8	89.9	86.1	91.1	87.3	90.5	85.8
AUC-ROC		66	94	90	96	92	97	93	96	92
MCC = 0.84 (on validation) and 0.75 (on test).										

The subsequent subsections show the results which were obtained after augmenting synthetic data to the original dataset to balance classes.

3.4.2 Different Domains Features' Contribution Analysis

To ensure the reliability and robustness of the method introduced in this work, we present experimental results based on widely-accepted performance metrics summarized in Table 3.4. To simplify the analysis, classification performance between time-domain, frequency-domain and combined features from both domains is graphically presented in Figure 3.6.

From Table 3.4 and Figure 3.6, it can be seen that the classification process taken



(a) Recall (b) Precision (c) F1-Score

(d) Accuracy (e) AUC-ROC

Figure 3.6. Performance metrics graphs

with time-domain features gave impressive validation and test results for both faithful and unfaithful customers classes. An experiment done with frequency-domain features alone showed improved results. The best results were obtained when all features from both domains were combined. For example, on validation, accuracy was 87.5%, which improved to 89.9%, and finally 91.1% when the experiment was done with time-domain features, frequency-domain features and all features from both domains respectively. The red trend line in Figure 3.6 graphs portrays significant improvement on results obtained from experiments done with time-domain features, frequency-domain features and all features from both domains. This improvement can be explained by a bar chart of predictors presented in order of their prominence shown in Figure 3.7, which has been produced through the mRMR scheme.

As shown by Figure 3.7 bar chart, there are more frequency-domain features to the left of the bar chart (i.e., features with the best scores) than time-domain features, with mean frequency achieving the highest predictor score. We confirmed the exactness of features' ranking through the mRMR scheme by doing classification tasks using top 3, middle 3 and bottom 3 features on the same network in Figure 3.5. Figure 3.8 bar chart shows classification accuracy and AUC-ROC results.

Comparing the results in Figure 3.8, we observed that accuracy and AUC-ROC

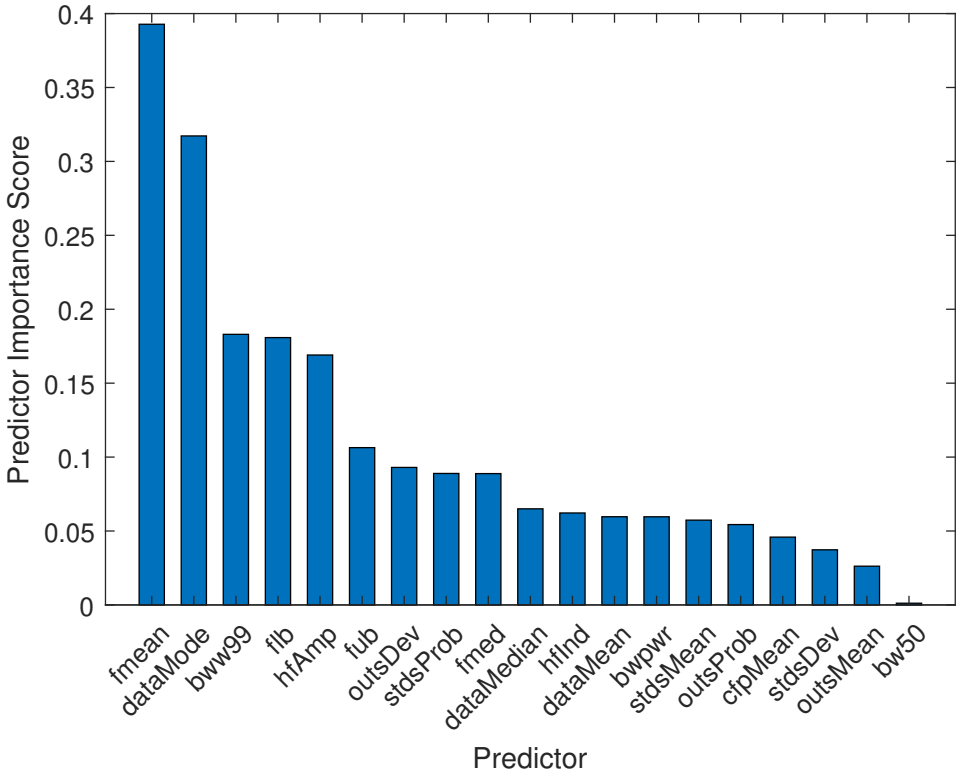


Figure 3.7. Features presented in order of their prominence

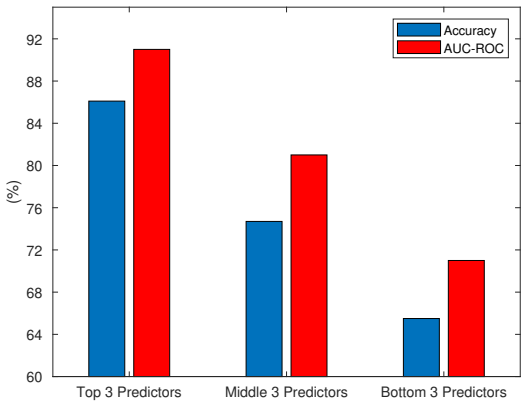


Figure 3.8. Classification results comparison of features ordered by mRMR scheme

are best for the top 3 features and worst for the bottom 3 features, as expected. MCC was determined on the last experiment when all features were combined. Its values were found to be 0.84 and 0.75 on validation and test respectively, which are closer to +1 than -1. AUC-ROC values were found to be 97% and 93%

on validation and test respectively. These results portray a satisfactory overall classification task.

3.4.3 Analysis of Components Reduction With PCA

When PCA was incorporated with the component reduction criterion of leaving enough components to explain 95% variance, seven components were left, having the following percentages contributions to total variance: 35.84%, 27.02%, 15.55%, 7.69%, 4.87%, 3.30% and 1.81%. Figure 3.9 shows 2-D biplots of original features contribution to each of the components in the principal components space.

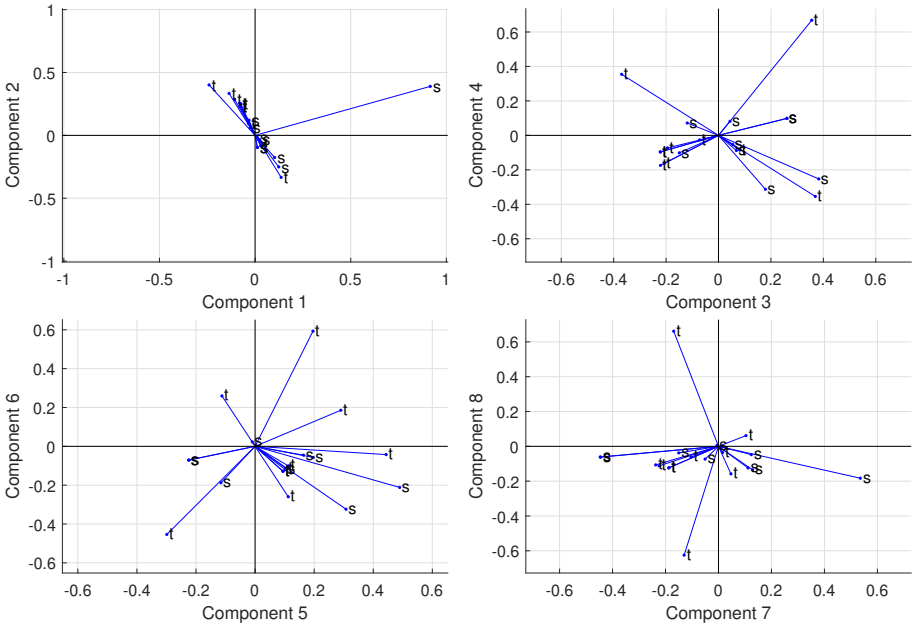


Figure 3.9. Graphical display of original features' contribution to principal components

Frequency-domain feature vectors are labelled with 's', while time-domain features are labelled with 't'. The contribution of each feature to the principal component is signified by that feature's vector direction and length. From Figure 3.9, we observed that frequency-domain features contributed more to principal components. This was also confirmed by features importance scores analysis shown by Figure 3.7 based on the mRMR scheme. The last two columns in Table

3.4 shows both validation and test results obtained after components reduction with PCA. We observed that with just seven principal components, we were able to achieve results very close to when no feature reduction criterion was used.

3.4.4 Hyperparameters Optimization Results

Following the hyperparameters optimization procedure stipulated in Section 3.3.3, Figure 3.10 shows observed objective function values vs optimization steps.

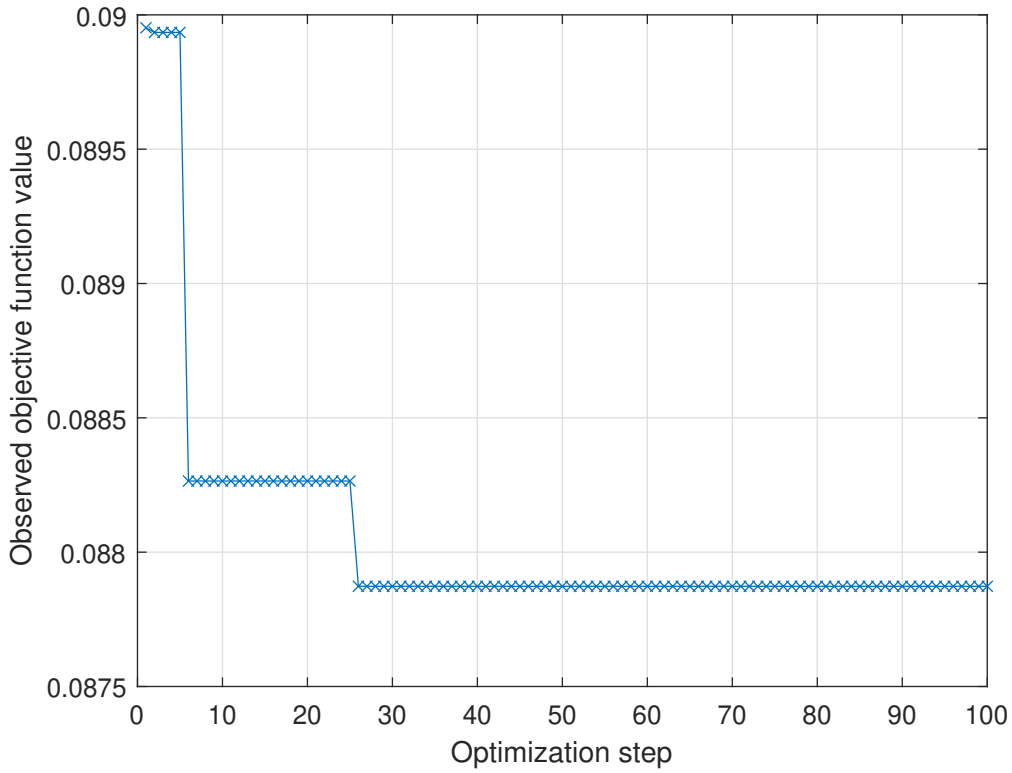


Figure 3.10. Objective function value vs optimization steps

The best hyperparameters combination was obtained at the 26^{th} optimization step and remained unchanged till the 100^{th} step. Their values are shown in Table 3.5.

An improved classification network architecture constructed with optimized hyperparameters achieved maximum validation and test accuracies of 91.8% and 88.1% respectively, which are 0.7% and 0.8% higher than an unoptimized architecture. The classifier obtained a maximum AUC-ROC value of 97%.

TABLE 3.5. Optimized hyperparameter values

Parameter	**Value**
Fully-connected Layers	[41 21]
Regularization strength	5.6882×10^{-7}
Activation function	Sigmoid

3.4.5 Key Parameters' Impact Analysis

Impact of Initial Learning Rate

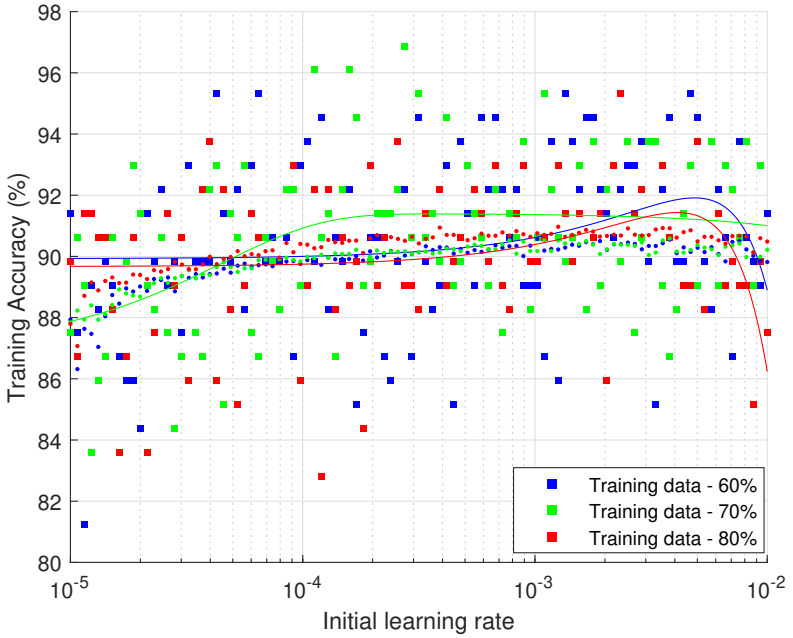
To determine the impact of the initial learning rate on training and validation accuracies, the initial learning rate was varied between 10^{-5} and 10^{-2} in 100 steps. Figure 3.11 shows scatter plots of the results with fitted curves to simplify analysis. For all tested training data portions, training and validation accuracies values were lowest for the lowest initial learning rates, with recorded values less than 90%.

Significant improvement in both accuracies was seen for initial learning rate values between 10^{-5} and 10^{-4}. Low values of accuracies were attained in this range because lower learning rates require higher training iterations (hence more training time) for the model to converge to good results, therefore accuracy was mainly limited by the limited number of epochs allowed to train the network. Beyond the initial learning rate of 10^{-4}, average training and validation accuracies improved beyond 90%.

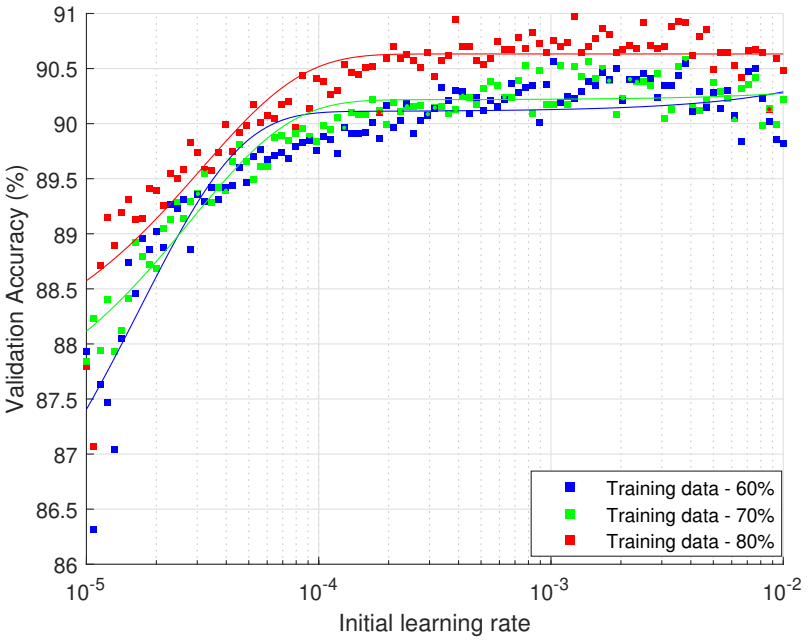
For all training data portions, the best accuracy values were obtained for initial learning rate values in the range $[10^{-3.7}, 10^{-2.5}]$. Both accuracies started dropping as the initial learning rate approached 10^{-2}. For optimal results, an initial learning rate in the range $[10^{-3.7}, 10^{-2.5}]$ is recommended for best accuracy.

Impact of Minibatch Size

To determine the impact of the minibatch size on the accuracy, the minibatch size was varied between 10^{1} and 10^{5} in 100 steps. We present training and validation accuracy versus minibatch size parameter plots in Figure 3.12. For all tested



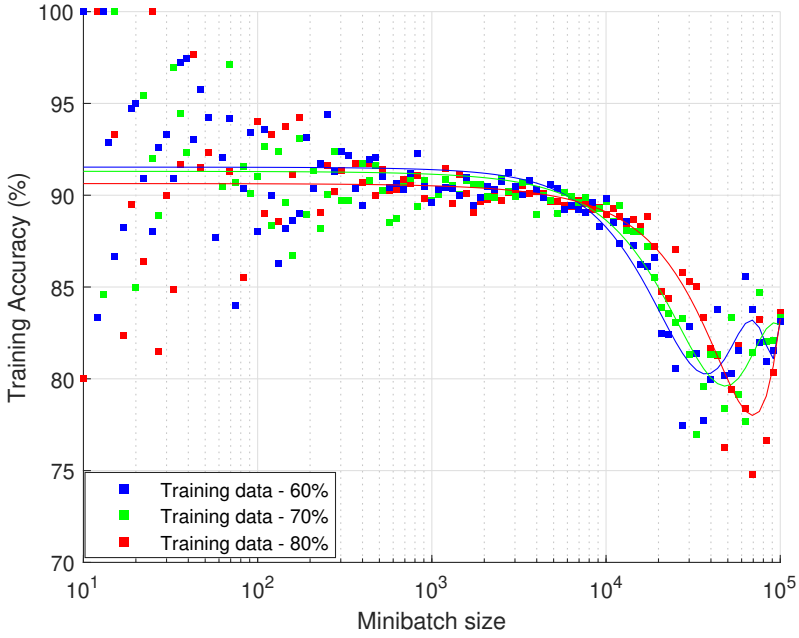
(a) Training



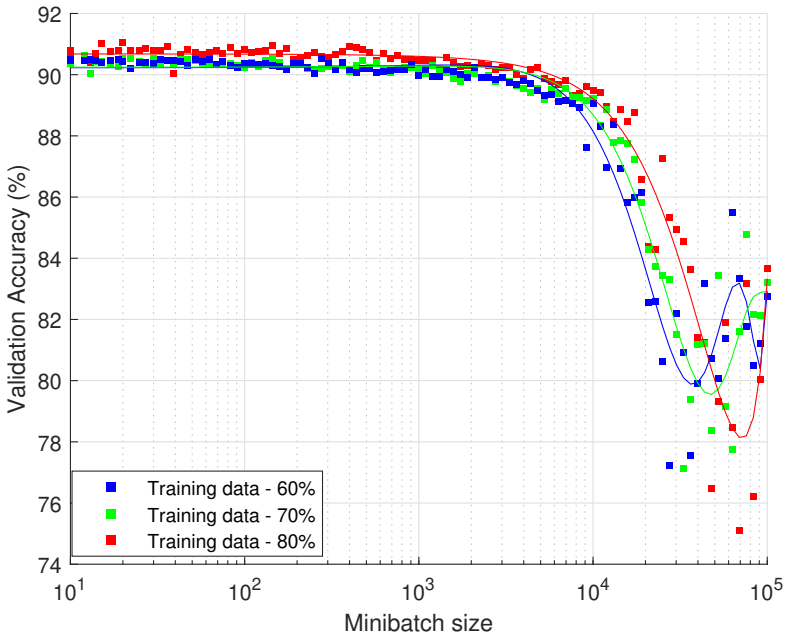
(b) Validation

Figure 3.11. Impact of varying initial learning rate on accuracy at different training ratios

training data portions, the training and validation accuracies averages were a little bit higher than 90% for minibatch size values less than 10^3.



(a) Training



(b) Validation

Figure 3.12. Impact of varying minibatch size on accuracy at different training ratios

For minibatch sizes closer to 10^1, the training accuracy varied significantly between 80% and 100% for each training task, however, this did not have an impact on validation as validation accuracy stayed the same just above 90%. Both train-

ing and validation accuracies declined drastically as minibatch size increased beyond 10^4. This is because as the value of the minibatch size increased, the model had to learn from increased data size, resulting in poor generalization. However, smaller minibatch size values required relatively much time to train a model. A minibatch size less than but closer to 10^3 is recommended to balance efficiency and generalization.

Impact of L2-regularization Parameter

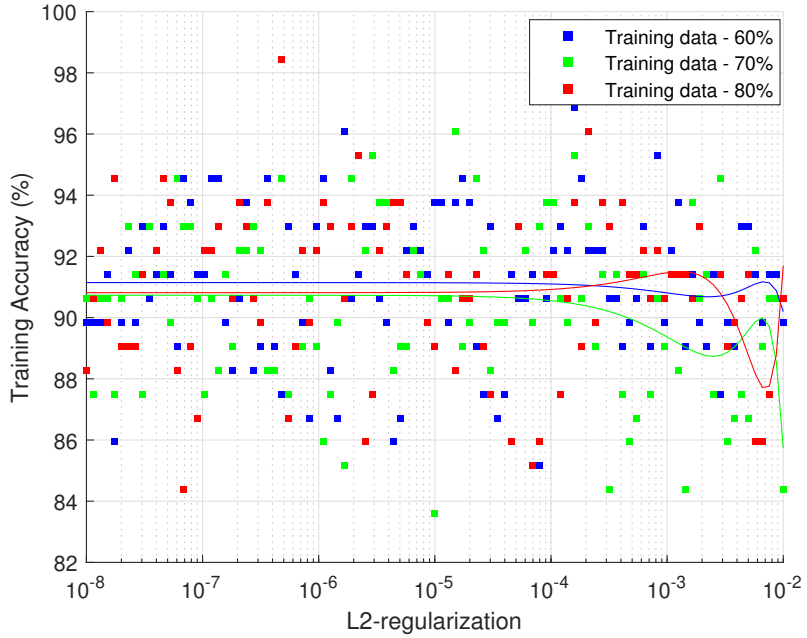
To determine the impact of the L2-regularization parameter on validation accuracy, the L2-regularization parameter was varied between 10^{-8} and 10^{-2} in 100 steps. Figure 3.13 shows the results. For all training data portions, training accuracy laid between 83% and 99%, with an average value at around 91% for L2-regularization parameter values in the range $[10^{-8}, 10^{-4})$.

Unstable average values of training accuracy were observed for L2-regularization parameter values $\geq 10^{-4}$. On the other hand, validation accuracy significantly decreased for L2-regularization parameter values $\geq 10^{-4}$. This may be caused by the fact that when the L2-regularization parameter $\geq 10^{-4}$, at each training iteration, a significantly large number of weights was left not updated, thereby making it hard for the model to converge to a good solution. Best results were obtained when L2-regularization parameter values were in the range $[10^{-8}, 10^{-4})$.

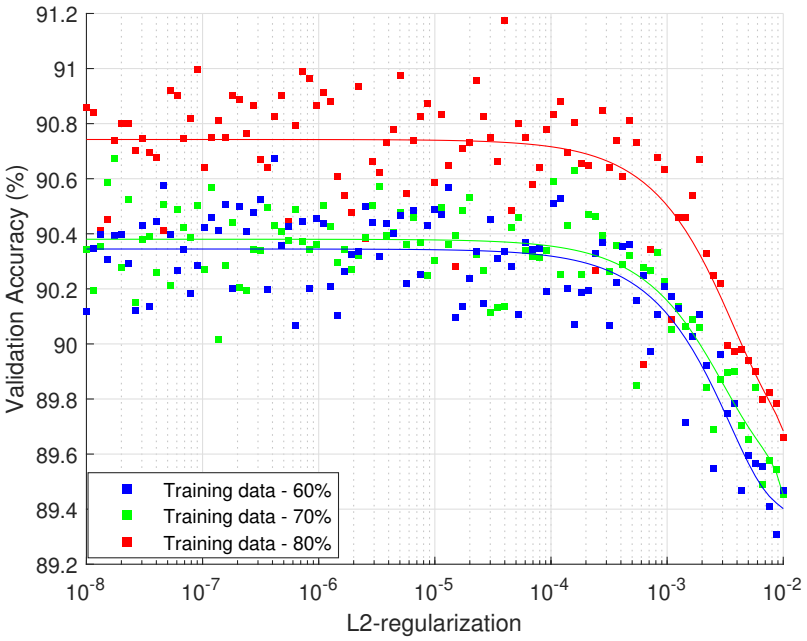
For all investigated parameters, the best validation accuracy was obtained for the 80% training data portion, followed by the 70% training data portion and lastly 60% training data portion. This shows that the more data is available for training the model, the more accurate the model becomes in detecting electricity theft.

3.4.6 Comparison With Existing Data-based Electricity Theft Detection Methods

Based on electricity customers consumption data, different data-driven methods have been used to tackle the electricity theft problem. Due to the scarcity of



(a) Training



(b) Validation

Figure 3.13. Impact of varying L2-regularization parameter on accuracy at different training ratios

datasets containing both faithful and unfaithful customers' consumption data, many methods have been evaluated on different uncommon datasets. In Table 3.6, we present analysis in the difference between our work and the recent works in

TABLE 3.6. Comparison with existing data-based electricity theft detection methods

Reference	Techniques/ Algorithms Used	Features used	Evaluation Dataset Details		Performance Evaluation
			Source	# of Customers	
[10]	SMOTE + KPCA + SVM.	Extracted from the original time-series data with KPCA.	SGCC	42372	Accuracy: 89% Precision: 85% Recall: 88%
[11]	Wide + deep CNN.	Wide and deep CNN used to learn features from the time-series data.	SGCC	42372	AUC: 79% Mean Average Precision (MAP): 96.9%
[50]	PCA + Calculation of anomaly score	PCs extracted from the original time-series data using PCA.	Irish smart meter data	5000	TPR: 90.9%
[69]	O-SVM + CS-SVM + OPF + C4.5 tree	Manually selected features from the original time-series data	Uruguayan electric power company	1504 3338 (two independent datasets)	Best accuracy: 86.2%
[70]	CNN + LSTM	CNN used to learn features from the time-series data.	SGCC	9956	Accuracy: 89% F1-score: 58.8%
[71]	LOF + k-means clustering	N/A	SGCC	3500	AUC: 81.5% MAP: 73.35%
[72]	SALM	AlexNet used to extract features from the original time-series data.	SGCC	42372	Accuracy: 90% AUC: 90.6%
[72]	GAN-NetBoost	GoogleNet used to extract features from the original time-series data.	SGCC	42372	Accuracy: 95% AUC: 96%
This work	**Feed forward DNN**	**Manually extracted time-domain and frequency-domain features.**	**SGCC**	**42372**	**Accuracy: 91.8% AUC: 97%**

the literature. For each work, dataset details are given. We look at the techniques and/or algorithms used, as well as features extracted from the data in respective methods.

For the four methods which used the same dataset as ours (References [10], [11], [72]), we compare the results in terms of AUC and accuracy percentages. We obtained AUC that is 1% higher than the best AUC in the benchmark and accuracy that is the second-best. The results show that our work is very competitive against other methods recently undertaken.

3.5 Conclusion

This chapter presented the detection of electricity theft in smart grids was investigated using time-domain and frequency-domain features in a DNN-based

classification approach. Isolated classification tasks based on the time-domain, frequency-domain and combined domains features were investigated on the same DNN network. Widely accepted performance metrics such as recall, precision, F1-score, accuracy, AUC-ROC and MCC were used to measure the performance of the model. We observed that classification done with frequency-domain features outperforms classification done with time-domain features, which in turn is outperformed by classification done with features from both domains. The classifier was able to achieve 87.3% accuracy and 93% AUC-ROC when tested.

We used PCA for feature reduction. With 7 out of 20 components used, the classifier was able to achieve 85.8% accuracy and 92% AUC-ROC when tested. We further analyzed individual features' contribution to the classification task and confirmed with the mRMR algorithm the importance of frequency-domain features over time-domain features towards a successful classification task. For better performance, a Bayesian optimizer was also used to optimize hyperparameters, which realized accuracy improvement close to 1%, on validation. Adam optimizer was incorporated and optimal values of key parameters were investigated. In comparison with other data-driven methods evaluated on the same dataset, we obtained 97% AUC which is 1% higher than the best AUC in existing works, and 91.8% accuracy, which is the second-best on the benchmark.

Chapter 4

Conclusion

4.1 Research Summary

This work was done with the aim to answer the following research question:

What level of performance can be realized by using time and frequency domains features to detect electricity theft in a DNN-based classification approach? An investigation is done to determine how mRMR can help in identifying individual features' contribution to successful theft detection, as well as how PCA can be used to reduce DNN input dimensionality to simplify the training process while maintaining electricity theft detection performance satisfactory.

To address the research question, a realistic dataset consisting of electricity consumption data of both classes of faithful and unfaithful customers was acquired. Customers' load profiles were analysed to decide the comprehensive time and frequency domains features that could be extracted and fed to the classifier. Before features could be extracted, the dataset had to be pre-processed to improve its quality to prepare it for the extraction of features. 19 time-domain and frequency-domain features are extracted and used in a DNN-based classification task. A DNN classifier was designed and tested with isolated classification tasks based on the time-domain, frequency-domain and combined domains features.

PCA was incorporated to reduce the dimension of DNN classifier input by keeping

enough principal components that explained 95% variance. Individual features' contributions to the classification task were investigated with the mRMR algorithm. The research further incorporated a Bayesian optimizer to tune hyperparameters so that the best performance could be achieved, and an Adam optimizer was used to investigate significant parameters' ranges of values that achieve the best classification results. Many widely accepted performance metrics were used to evaluate the performance of the model, and the model performance was compared with other recent models' performance in the literature. Performance comparison was made with the four models which utilized the same dataset used in this work.

4.2 Future Work Recommendations

In order to prepare data for classification, PCHIP was used for data interpolation and the data imbalance problem was solved by generating synthetic data. Signal processing techniques such as wavelets, space-time, etc. may be considered in the data pre-processing stage. One-class classification algorithms may also be considered for imbalanced class datasets such as the one used in this work. Since the method used here is based on consumption patterns of SGCC customers, it can further be validated against datasets from different areas once available, to ensure its applicability anywhere. This method can be improved to detect real-time electricity theft.

4.3 Conclusion

This work presented a novel electricity theft detection scheme based on fully connected feed-forward DNN classification using time-domain and frequency-domain features. Based on recall, precision, F1-score, accuracy and AUC-ROC, it was observed that classification done with frequency-domain features outperforms classification done with time-domain features, which in turn is outperformed by classification done with features from both domains. The importance of frequency-domain features over time-domain features towards a successful classification task

was also confirmed with both PCA and mRMR. The manually tuned classifier was able to achieve 87.3% accuracy and 93% AUC-ROC when tested.

With DNN input dimensionality reduction using PCA, just 7 out of 19 principal components were found to explain the acceptable variance of 95%. The classifier achieved 85.8% accuracy and 92% AUC-ROC when tested, which are within just a 2% margin close to results when no PCA was used. With Bayesian optimized hyperparameters, accuracy improvement close to 1% was realized. Accuracy and AUC-ROC metrics were used to compare this work with other data-driven works in the literature which have been evaluated on the same SGCC dataset. This work showed 1% improvement on AUC obtained by the best model and second-best accuracy. The method used here utilized consumption data patterns. Apart from its relevance in power distribution networks, it can also be used in anomaly detection applications in any field.

Appendix A

Loss Function

A.1 Chapter Summary

When the classification model is trained, the goal of the classifier is to minimize the loss function. This chapter presents the results of binary cross-entropy loss minimization. The results of the training of the DNN model in Figure 3.5 with all extracted features are presented here as an example.

A.2 Binary Cross-entropy Loss Minimization

In this work, the training of a fully connected feed-forward DNN aimed to minimize the binary cross-entropy loss function. To get the best performance results, the training iterations were planned in such a way that the last training iteration is attained when the cross-entropy loss function is at the minimum value. 1000 iterations were used.

Figure A.1 shows cross-entropy loss versus training iterations when the network shown by Figure 3.5 was trained with all features from time-domain and frequency-domain. The loss value drastically declined from 0.35 to 0.12 in the first 50 iterations. It gradually went down from the 50^{th} iteration and became stable just below 0.11 until the last iteration.

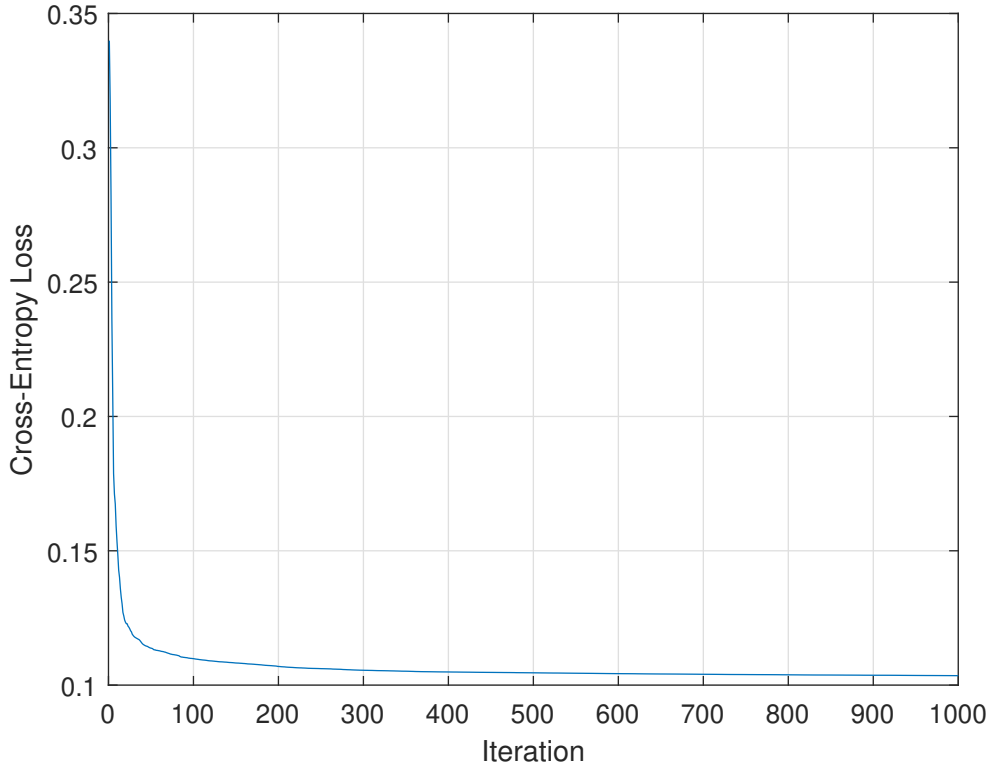


Figure A.1. Cross-entropy loss vs training iterations

Appendix B

Confusion Matrix

B.1 Chapter Summary

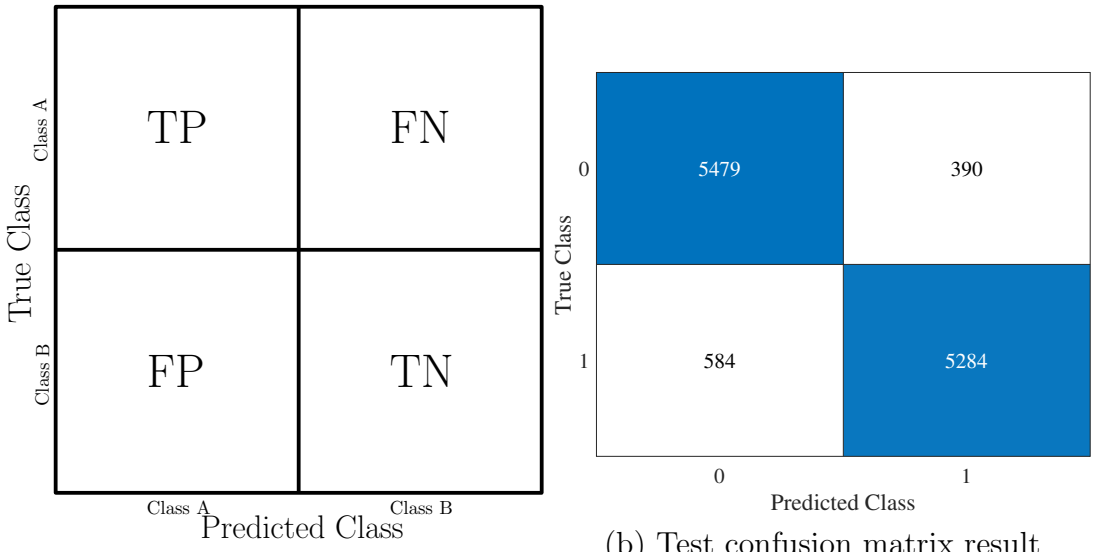
In this chapter, a confusion matrix of a binary classifier is presented. A typical structure of a confusion matrix is presented, followed by confusion matrix test results obtained from training the DNN model in Figure 3.5 with all extracted features.

B.2 Confusion Matrix Results

Confusion matrices played an important part as they contained information about actual and predicted classifications of a classifier. Figure B.1a shows a typical structure of a confusion matrix of a binary classifier. Figure B.1b shows test results in the form of a confusion matrix obtained from classification done with all features from time-domain and frequency-domain as an example.

Given one of the two classes in a binary classification problem, [87] describes confusion matrix elements as follows:

- TP: examples correctly recognized as belonging to a class.
- TN: examples correctly recognized as not belonging to a class.



(a) Binary classifier confusion matrix structure

(b) Test confusion matrix result

Figure B.1. Confusion matrices

- FP: examples incorrectly recognized as belonging to a class.
- FN: examples incorrectly recognized as not belonging to a class.

In Figure B.1b, examples that were correctly classified are highlighted in blue. Examples that were not correctly classified are not highlighted.

References

[1] M. Waite, E. Cohen, H. Torbey, M. Piccirilli, Y. Tian, and V. Modi, "Global trends in urban electricity demands for cooling and heating," *Energy*, vol. 127, pp. 786–802, 2017.

[2] T. Vijayapriya and D. P. Kothari, "Smart grid: An overview," *Smart Grid and Renewable Energy*, vol. 2, no. 4, p. 305, 2011.

[3] G. Florea, O. Chenaru, R. Dobrescu, and D. Popescu, "Evolution from power grid to smart grid: Design challenges," in *2015 19th International Conference on System Theory, Control and Computing (ICSTCC)*, IEEE, 2015, pp. 912–916.

[4] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—the new and improved power grid: A survey," *IEEE communications surveys & tutorials*, vol. 14, no. 4, pp. 944–980, 2011.

[5] J. Navani, N. Sharma, and S. Sapra, "Technical and non-technical losses in power system and its economic consequence in indian economy," *International Journal of Electronics and Computer Science Engineering*, vol. 1, no. 2, pp. 757–761, 2012.

[6] P. Massaferro, H. Marichal, M. Di Martino, F. Santomauro, J. P. Kosut, and A. Fernandez, "Improving electricity non technical losses detection including neighborhood information," in *2018 IEEE Power & Energy Society General Meeting (PESGM)*, IEEE, 2018, pp. 1–5.

[7] S. Foster, *Non-technical losses: A $96 billion global opportunity for electrical utilities*, Available at https://energycentral.com/c/pip/non-technical-losses-96-billion-global-opportunity-electrical-utilities (2021/11/02).

[8] Q. Louw and P. Bokoro, "An alternative technique for the detection and mitigation of electricity theft in South Africa," *SAIEE Africa Research Journal*, vol. 110, no. 4, pp. 209–216, 2019.

[9] P. Pickering, *E-meters offer multiple ways to combat electricity theft and tampering*, Available at https://www.electronicdesign.com/technologies/meters (2021/11/01).

[10] M. Anwar, N. Javaid, A. Khalid, M. Imran, and M. Shoaib, "Electricity theft detection using pipeline in machine learning," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, IEEE, 2020, pp. 2138–2142.

[11] Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou, "Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1606–1615, 2017.

[12] M. Ismail, M. Shahin, M. F. Shaaban, E. Serpedin, and K. Qaraqe, "Efficient detection of electricity theft cyber attacks in AMI networks," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2018, pp. 1–6.

[13] A. Maamar and K. Benahmed, "Machine learning techniques for energy theft detection in AMI," in *Proceedings of the 2018 International Conference on Software Engineering and Information Management*, 2018, pp. 57–62.

[14] A. Jindal, A. Schaeffer-Filho, A. K. Marnerides, P. Smith, A. Mauthe, and L. Granville, "Tackling energy theft in smart grids through data-driven analysis," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, IEEE, 2020, pp. 410–414.

[15] I. Diahovchenko, M. Kolcun, Z. Čonka, V. Savkiv, and R. Mykhailyshyn, "Progress and challenges in smart grids: Distributed generation, smart metering, energy storage and smart loads," *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, vol. 44, no. 4, pp. 1319–1333, 2020.

[16] M. Jaganmohan, *Global smart grid market size by region 2017-2023*, Available at https://www.statista.com/statistics/246154/global-smart-grid-market-size-by-region/ (2022/03/03).

[17] Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou, *Electricity theft detection*, Available at https://github.com/henryRDlab/ElectricityTheftDetection (2021/09/30).

[18] S. McLaughlin, B. Holbert, A. Fawaz, R. Berthier, and S. Zonouz, "A multi-sensor energy theft detection framework for advanced metering infrastructures," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 7, pp. 1319–1330, 2013.

[19] Z. M. Fadlullah, N. Kato, R. Lu, X. Shen, and Y. Nozaki, "Toward secure targeted broadcast in smart grid," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 150–156, 2012.

[20] S. McLaughlin, D. Podkuiko, S. Miadzvezhanka, A. Delozier, and P. McDaniel, "Multi-vendor penetration testing in the advanced metering infrastructure," in *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010, pp. 107–116.

[21] K. Phil, *MATLAB deep learning: With machine learning, neural networks and artificial intelligence*. Seoul, Soul-t'ukpyolsi, Korea (Republic of): Apress, New York, 2017.

[22] S. Notley and M. Magdon-Ismail, "Examining the use of neural networks for feature extraction: A comparative analysis using deep learning, support vector machines, and k-nearest neighbor classifiers," *arXiv e-prints*, arXiv–1805, 2018.

[23] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks," *arXiv preprint arXiv:1710.02913*, vol. 9, 2017.

[24] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing*, 2nd ed. Prentice Hall, 2020.

[25] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA transactions on Signal and Information Processing*, vol. 3, 2014.

[26] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[27] PARISlab@UCLA, *Training an artificial neural network with MATLAB – Machine learning for engineers*, Available at https://youtu.be/xOzh6PMk21I (2021/07/19).

[28] J. Heaton, *Introduction to Neural Networks with Java*, 2nd ed. 1734 Clarkson Rd., Chesterfield: Heaton Research, Inc, 2008.

[29] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018.

[30] J. Zhang, "Gradient descent based optimization algorithms for deep learning models training," *arXiv e-prints*, arXiv–1903, 2019.

[31] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese control and decision conference (CCDC)*, IEEE, 2018, pp. 1836–1841.

[32] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[33] J. Feng and S. Lu, "Performance analysis of various activation functions in artificial neural networks," in *Journal of physics: conference series*, IOP Publishing, vol. 1237, 2019, p. 022 030.

[34] P. Dangeti, *Statistics for machine learning*. Birmingham – Mumbai: Packt Publishing Ltd, 2017.

[35] S. H. Haji and A. M. Abdulazeez, "Comparison of optimization techniques based on gradient descent algorithm: A review," *PalArch's Journal of Archaeology of Egypt/Egyptology*, vol. 18, no. 4, pp. 2715–2743, 2021.

[36] W. Daelemans, B. Goethals, and K. Morik, *Proceedings of the 2008th european conference on machine learning and knowledge discovery in databases-volume part i*, 2008.

[37] D. P. Kingma and J. Ba, *Adam: A method for stochastic opoimization*, 2017.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv e-prints*, arXiv–1609, 2016.

[39] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[40] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on Bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[41] P. I. Frazier, "A tutorial on Bayesian optimization," *stat*, vol. 1050, p. 8, 2018.

[42] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv e-prints*, arXiv–1012, 2010.

[43] A. Ghodsi, "Dimensionality reduction: A short tutorial," *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada*, vol. 37, no. 38, p. 2006, 2006.

[44] J. Shlens, "A tutorial on principal component analysis," *arXiv e-prints*, arXiv–1404, 2014.

[45] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20 150 202, 2016.

[46] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[47] R. Bro and A. K. Smilde, "Principal component analysis," *Analytical methods*, vol. 6, no. 9, pp. 2812–2831, 2014.

[48] M. Richardson, "Principal component analysis," 2009.

[49] R. Reris and J. P. Brooks, "Principal component analysis and optimization: A tutorial," 2015.

[50] S. K. Singh, R. Bose, and A. Joshi, "PCA based electricity theft detection in advanced metering infrastructure," in *2017 7th International Conference on Power Systems (ICPS)*, IEEE, 2017, pp. 441–445.

[51] H. Abdi, "Singular value decomposition (SVD) and generalized singular value decomposition," *Encyclopedia of measurement and statistics*, pp. 907–912, 2007.

[52] K. Baker, "Singular value decomposition tutorial," *The Ohio State University*, vol. 24, 2005.

[53] E. W. Weisstein, *Eigenvalue.* Available at https://mathworld.wolfram.com/Eigenvalue.html (2022/02/20).

[54] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[55] M. Toğaçar, B. Ergen, and Z. Cömert, "Detection of lung cancer on chest ct images using minimum redundancy maximum relevance feature selection method with convolutional neural networks," *Biocybernetics and Biomedical Engineering*, vol. 40, no. 1, pp. 23–39, 2020.

[56] T. Liu, L. Hu, C. Ma, Z.-Y. Wang, and H.-L. Chen, "A fast approach for detection of erythemato-squamous diseases based on extreme learning machine with maximum relevance minimum redundancy feature selection," *International Journal of Systems Science*, vol. 46, no. 5, pp. 919–931, 2015.

[57] M. Billah and S. Waheed, "Minimum redundancy maximum relevance (mRMR) based feature selection from endoscopic images for automatic gastrointestinal polyp detection," *Multimedia Tools and Applications*, vol. 79, no. 33, pp. 23 633–23 643, 2020.

[58] C. Ding and H. Peng, "Minimum redundancy feature selection from microarray gene expression data," in *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, 2003, pp. 523–528. DOI: 10.1109/CSB.2003.1227396.

[59] D. O. Dike, U. A. Obiora, E. C. Nwokorie, and B. C. Dike, "Minimizing household electricity theft in Nigeria using GSM based prepaid meter," *American Journal of Engineering Research (AJER) e-ISSN*, vol. 23200847, pp. 2320–0936, 2015.

[60] P. Dhokane, M. Sanap, P. Anpat, J. Ghuge, and P. Talole, "Power theft detection & initimate energy meter information through SMS with auto power cut off," *International Journal of Current Research in Embedded System & VLSI Technology [ISSN: 2581-5105 (online)]*, vol. 2, no. 1, 2017.

[61] S. B. Yousuf, M. Jamil, M. Z. ur Rehman, A. Hassan, and S. O. G. Syed, "Prototype development to detect electric theft using PIC18F452 microcontroller," *Indian Journal of Science and Technology*, vol. 9, no. 46, 2016.

[62] K. Dineshkumar, P. Ramanathan, and S. Ramasamy, "Development of arm processor based electricity theft control system using GSM network," in *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, IEEE, 2015, pp. 1–6.

[63] S. Ngamchuen and C. Pirak, "Smart anti-tampering algorithm design for single phase smart meter applied to AMI systems," in *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, IEEE, 2013, pp. 1–6.

[64] B. Khoo and Y. Cheng, "Using RFID for anti-theft in a chinese electrical supply company: A cost-benefit analysis," in *2011 Wireless Telecommunications Symposium (WTS)*, IEEE, 2011, pp. 1–6.

[65] J. Astronomo, M. D. Dayrit, C. Edjic, and E. R. T. Regidor, "Development of electricity theft detector with GSM module and alarm system," in *2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, IEEE, 2020, pp. 1–5.

[66] P. Jokar, N. Arianpoo, and V. C. Leung, "Electricity theft detection in AMI using customers' consumption patterns," *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 216–226, 2015.

[67] W. Han and Y. Xiao, "A novel detector to detect colluded non-technical loss frauds in smart grid," *Computer Networks*, vol. 117, pp. 19–31, 2017.

[68] S. Sahoo, D. Nikovski, T. Muso, and K. Tsuru, "Electricity theft detection using smart meter data," in *2015 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, IEEE, 2015, pp. 1–5.

[69] M. Di Martino, F. Decia, J. Molinelli, and A. Fernández, "Improving electric fraud detection using class imbalance strategies.," in *ICPRAM (2)*, 2012, pp. 135–141.

[70] M. Hasan, R. N. Toma, A.-A. Nahid, M. Islam, J.-M. Kim, *et al.*, "Electricity theft detection in smart grid systems: A CNN-LSTM based approach," *Energies*, vol. 12, no. 17, p. 3310, 2019.

[71] Y. Peng, Y. Yang, Y. Xu, *et al.*, "Electricity theft detection in AMI based on clustering and local outlier factor," *IEEE Access*, vol. 9, pp. 107 250–107 259, 2021.

[72] A. Aldegheishem, M. Anwar, N. Javaid, N. Alrajeh, M. Shafiq, and H. Ahmed, "Towards sustainable energy efficiency with intelligent electricity theft detection in smart grids emphasising enhanced neural networks," *IEEE Access*, vol. 9, pp. 25 036–25 061, 2021.

[73] C. Moler, *Splines and pchips*, Available at https://blogs.mathworks.com/cleve/2012/07/16/splines-and-pchips/ (2021/09/05).

[74] G. Dong and H. Liu, *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[75] B. D. Fulcher and N. S. Jones, "Highly comparative feature-based time-series classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3026–3037, 2014.

[76] Codecademy, *Normalization*, Available at https://www.codecademy.com/articles/normalization (2021/09/05).

[77] M. Fitz, *Fundamentals of Communication Systems*. United States of America: McGraw-Hill, 2007.

[78] B. Vega-Márquez, I. Nepomuceno-Chamorro, N. Jurado-Campos, and C. Rubio-Escudero, "Deep learning techniques to improve the performance of olive oil classification," *Frontiers in chemistry*, vol. 7, p. 929, 2020.

[79] K. T. Chui, D. C. L. Fung, M. D. Lytras, and T. M. Lam, "Predicting at-risk university students in a virtual learning environment via a machine learning algorithm," *Computers in Human Behavior*, vol. 107, p. 105 584, 2020.

[80] K. E. Koech, *Cross-entropy loss function*, Available at https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e (2022/02/07).

[81] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[82] A. F. Clark and C. Clark, "Performance characterization in computer vision a tutorial," *) N (Eds.): FBook performance characterization in computer vision a tutorial/(Citeseer, 1999, edn.)*, 1999.

[83] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[84] Google-Developers, *Classification: Accuracy*, Available at https://developers.google.com/machine-learning/crash-course/classification/accuracy (2021/09/06).

[85] J. Brownlee, *Classification accuracy is not enough: More performance measures you can use*, Available at https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/ (2021/09/06).

[86] A. Bhandari, *AUC-ROC curve in machine learning clearly explained*, Available at https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/ (2021/09/06).

[87] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[88] M. K. Uçar, M. Nour, H. Sindi, and K. Polat, "The effect of training and testing process on machine learning in biomedical datasets," *Mathematical Problems in Engineering*, vol. 2020, 2020.