UNIVERSITY of STIRLING

OPTIMISING WLANS POWER SAVING: CONTEXT-AWARE LISTEN INTERVAL

AHMED SAEED

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Doctor of Philosophy

Institute of Computing Science and Mathematics

University of Stirling

September 2023

DECLARATION OF THE AUTHORSHIP

I hereby declare that this thesis,

"Optimising WLANs Power Saving: Context-Aware Listen Interval"

To the best of my knowledge this is entirely my work, and where any material points to the ideas of others, it is thoroughly cited and referenced with appropriate acknowledgements given. Also, it has not been submitted for examination for any other degree at this university or any other learning institutions.

Ahmed Saeed

September 2023

ABSTRACT

Energy is a vital resource in wireless computing systems. Despite the increasing popularity of Wireless Local Area Networks (WLANs), one of the most important outstanding issues remains the power consumption caused by Wireless Network Interface Controller (WNIC). To save this energy and reduce the overall power consumption of wireless devices, a number of power saving approaches have been devised including Static Power Save Mode (SPSM), Adaptive PSM (APSM), and Smart Adaptive PSM (SAPSM). However, the existing literature has highlighted several issues and limitations in regards to their power consumption and performance degradation, warranting the need for further enhancements.

This thesis proposes a novel Context-Aware Listen Interval (CALI), in which the wireless network interface, with the aid of a Machine Learning (ML) classification model, sleeps and awakes based on the level of network activity of each application. We focused on the network activity of a single smartphone application while ignoring the network activity of applications running simultaneously.

We introduced a context-aware network traffic classification approach based on ML classifiers to classify the network traffic of wireless devices in WLANs. Smartphone applications' network traffic reflecting a diverse array of network behaviour and interactions were used as contextual inputs for training ML classifiers of output traffic, constructing an ML classification model. A real-world dataset is constructed, based on nine smartphone applications' network traffic, this is used firstly to evaluate the performance of five ML classifiers using cross-validation, followed by conducting extensive experimentation to assess the generalisation capacity of the selected classifiers on unseen testing data. The experimental results further validated the practical application of the selected ML classifiers and indicated that ML classifiers can be usefully employed for classifying the network traffic of smartphone applications based on different levels of behaviour and interaction.

Furthermore, to optimise the sleep and awake cycles of the WNIC in accordance with the smartphone applications' network activity. Four CALI power saving modes were

developed based on the classified output traffic. Hence, the ML classification model classifies the new unseen samples into one of the classes, and the WNIC will be adjusted to operate into one of CALI power saving modes. In addition, the performance of CALI's power saving modes were evaluated by comparing the levels of energy consumption with existing benchmark power saving approaches using three varied sets of energy parameters. The experimental results show that CALI consumes up to 75% less power when compared to the currently deployed power saving mechanism on the latest generation of smartphones, and up to 14% less energy when compared to SAPSM power saving approach, which also employs an ML classifier.

Dedicated to

My loving Parents, Brothers, Sister, and their children

For their love, prayers, encouragement, and support. I love you all!

ACKNOWLEDGEMENTS

In the name of Allah, the most gracious and the most merciful. First and foremost, I am extremely thankful to Almighty Allah for his blessings and for giving me the strength and knowledge to carry out this research, without which none of my work would have been possible.

My sincerest gratitude and deepest appreciation goes to my principal supervisor, **Dr. Mario Kolberg**, an exceptional mentor, for his continuous and unlimited guidance, encouragement, motivation, and support throughout my PhD study, without which this thesis would not have been produced in the current form. His insightful comments, frequent availability, constant feedback, and suggestions from him have provided valuable guidance in various phases throughout this research journey. Working with him has been one of the most rewarding experiences of my professional life. I am very proud and honoured to be one of his PhD students.

I would also like to express my deepest thanks to my parents, **Dr. Saeed Inayatullah** and **Mrs. Nargis Sarwar**, for their unconditional love, kindness, and encouragement to whom this work is dedicated. They supported me unconditionally throughout my education and dedicated their entire lives to my success.

Finally, my greatest thanks go to my brothers and sister, Mr. Muhammad, Mrs. Sobia, Mr. Hammad, Mr. Hamzah, and their children Miss Sara, Miss Raudah, Miss Eman, and Mr Mustafa for their love, support, and encouragement throughout my study. I wish them health, happiness and just everything their heart desires.

LIST OF PUBLICATIONS

During the period of this research, the following journal papers have been published:

1 A. Saeed and M. Kolberg, "Towards Optimizing WLANs Power Saving: Context-Aware Listen Interval," in IEEE Access, vol. 9, pp. 141513-141523, 2021, doi: 10.1109/ACCESS.2021.3120348. (Materials from this journal paper are included within this thesis in Chapter 2, 3, 4, and 7).

2 A. Saeed and M. Kolberg, "Towards Optimizing WLANs Power Saving: Novel Context-Aware Network Traffic Classification Based on a Machine Learning Approach," in IEEE Access, vol. 7, pp. 3122-3135, 2019, doi: 10.1109/ACCESS.2018.2888813. (Materials from this journal paper are included within this thesis in Chapter 2, 3, 4, and 7).

CONTENTS

LIST OF ACRONYMS

Acronyms	Meaning of Acronyms
AAA	The Authentication, Authentication, and Accounting
Ack	Acknowledgment
ADUs	Application Data Units
AE	Auto Encoders
AID	Association ID
ANNs	Artificial Neural Networks
AP	Access Point
APSD	Automatic Power Save Delivery
APSM	Adaptive Power Save Mode / Adaptive PSM
CALI	Context-Aware Listen Interval
CBFS	Consistency Based Feature Selection
CNN	Convolutional Neural Networks
C-PSM	Centralized-Power Save Mode / Centralized-PSM
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSV	Comma-Separated-Values
CTS	Clear to Send
DBNs	Deep Belief Networks
DBSCAN	Density-based spatial clustering of applications with noise
DCF	Distributed Coordination Function
DL	Deep Learning
DLI	Dynamic Listen Interval
DPI	Deep Packet Inspection
DSL	Digital Subscriber Line
EM	Expectation-Maximisation clustering
E-MiLi	Energy-Minimizing idle Listening
FCFS	First Come First Serve
FN	False Negatives
FNR	False Negative Rate
FP	False Positives

FPR	False Positive Rate
GPU	Graphics Processing Unit
HD	High Definition
HPSM	Harmonious Power Saving Mechanism
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IBSS	Infrastructure Basic Service Set
ID3	Iterative Dichotomiser 3
IEEE	Institute of Electrical and Electronics Engineers
IGFS	Information Gain Feature Selection
IoT	Internet of Things
IP	Internet Protocol
KKT	Karush–Kuhn–Tucker
KNN	K-Nearest Neighbour
LAWS	Load-Aware Wakeup Scheduling
LDA	Linear Discriminant Analysis
MAC	Media Access Control
ML	Machine Learning
MLP	Multilayer Perceptron
MRMR	Maximum Relevance Minimum Redundancy
ms	Millisecond
NAPman	Network Assistant Power Management
NAV	Network Allocation Vector
NED	Network Description
NLP	Natural Language processing
NS	Network Simulator
NSC	New Star Cricket
NSS	New Star Soccer
OMNeT++	Objective Modular Network Testbed in C++
OPSM	Opportunistic Power Saving Mode
OS	Operating system
OTcl	Object Extension Tool command language

OvA	One versus All
OvO	One versus One
P2P	Peer to Peer
PHY-Assisted Power Saving	Physical layer Assisted Power Saving
PR	Precision Recall
PSM	Power Save Mode
PSMP	Power Save Multi Poll
PS-Poll	Power Save Poll
QAP	Quality Access Point
QoS	Quality of Service
RAM	Random Access Memory
RBF	Radial Basis Function
RF	Radio frequency
RLDA	Regularised Linear Discriminant Analysis
RNNs	Recurrent Neural Networks
ROC	Receiver Operator characteristics
RTS	Request to Send
S-APSD	Scheduled Adaptive Power Save Mode / Scheduled S-APSD
SAPSM	Smart Adaptive Power Save Mode / Smart Adaptive PSM
SiFi	A silence prediction-based Wi-Fi energy adaptation approach
SLA	Service Level Agreements
SMPS	Spatial Multiplexing Power Save
SMTP	Simple Mail Transfer Protocol
SYN	Synchronise
SOFA	A Sleep-Optimal Fair-Attention
SPSM	Static Power Save Mode
S-PSMP	Scheduled Power Save Multi Poll
SVM	Support Vector Machine
TCP	Transmission Control Protocol

TDMA	Time Division Multiple Access
TIM	Traffic Indication Map
TL	Transfer Learning
TN	True Negatives
TNR	True Negative Rate
TP	True Positives
TPR	True Positive Rate
TWT	Target Wake Time
U-APSD	Unscheduled Adaptive Power Save Mode / Unscheduled U-APSD
UDP	User Datagram Protocol
U-PSMP	Unscheduled Power Save Multi Poll
VoD	Video on Demand
VoIP	Voice over Internet Protocol
Wi-Fi	Wireless Fidelity
Wi-Fi Alliance	wireless industry organisation that exists to promote wireless technologies and interoperability
WiMAX	Worldwide Interoperability for Microwave Access
WLANs	Wireless Local Area Networks
WNIC	Wireless Network Interface Controller
WSNs	Wireless Sensor Networks
µPM	Micro-Power Management

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 Motivation

IEEE 802.11 Wireless Local Area Networks (WLANs), commercially known as Wi-Fi, are in pervasive deployment and considered one of the most rapidly growing technologies in the world that play an integral role in our lives [44]. According to a recent study from the Wi-Fi Alliance, there will be more than 16.4 billion Wi-Fi devices, including personal computers, laptops, smartphones, tablets, television and so on in use by the end of 2021 [45].

In an infrastructure-based WLANs, wireless devices are equipped with the Wireless Network Interface Controller (WNIC). WNIC allows wireless devices to share, communicate and access information wirelessly through an Access Point (AP) [46].

Energy is a vital resource in wireless computing systems, and despite the rapidly growing popularity of WLANs, one of the most important outstanding issues remains the power consumption caused by WNIC during data transferring between a wireless device and an AP. The high level of power consumption during the communication of WNIC directly affects the battery life of a wireless device, if is not connected to a power outlet [47, 48].

The 802.11 standard defines the Static Power Save Mode (SPSM) to reduce the amount of energy consumed by WNIC. In SPSM, a wireless device conserves energy by allowing the WNIC to sleep and waking up periodically to receive the buffered packets from the AP [1].

However, the SPSM suffers from latency issues. These occur firstly, when a wireless device generates the Power Save Poll (PS-Poll) frames to retrieve the buffered packets from the AP, and secondly, when a delay of about 100-300ms is introduced when the WNIC is off during the beacon intervals, but buffered packets are available at the AP. These issues affect the performance of both real-time applications, for example VoIP, and interactive applications, such as web browsers [4, 5, 6].

The Adaptive PSM (APSM) has been deployed within the latest generation of mobile devices to overcome the latency related issues associated with SPSM. In APSM, by default, the WNIC remains in SPSM and switches into awake mode based on a network activity threshold [7, 8]. But without considering the priority level of the network traffic of applications, this leads to unnecessary wakeups [5, 10]. Moreover, the WNIC remains in awake mode for an idle timeout period before fully switching back to SPSM [11]. Examples of currently existing smartphones that employ APSM which is based on SPSM are: Samsung Galaxy S10, Xiaomi Mi 10, ASUS ROG, and ROG II [168].

To eliminate the issue of the threshold mechanism built-in APSM, Smart Adaptive PSM (SAPSM) was proposed in [10]. Unlike SPSM and APSM which have been commercially deployed, SAPSM is still a research topic. SAPSM labels each network-based application of smartphone into two sets of priorities; high and low, with aid the of a Machine Learning (ML) classifier. SAPSM replaced the threshold mechanism of APSM with a set of two priorities, high and low. Consequently, for applications set as high priority, the WNIC will be adaptively switched into awake mode, and stays in the SPSM with applications set as low priority conserving energy.

However, no further priority levels or modes have been proposed in this work to cater for applications with different patterns of network activity: including the least levels of network interactivity, that receive network updates after

longer periods of time, secondly, applications with intermittent network interactions, and finally, for applications with buffering capabilities. Instead, SAPSM operates the WNIC in SPSM for all low priority applications.

This does not achieve the full potential of a methodology that considers optimising the sleep and awake cycles of the WNIC more closely in accordance with the smartphone applications' network traffic reflecting a diverse array of network behaviour and interactions.

1.2 Aims and Objectives

The aim of this thesis is to develop a power saving framework that optimises the sleep and awake cycles of the WNIC using Machine Learning (ML) techniques in accordance with smartphone applications' network traffic. To achieve this aim, the following research objectives have been determined.

- Identify and construct a real-world dataset based on a varied range of smartphone applications' network traffic depicting different types of network behaviour and interaction.
- Train ML classifiers to learn mapping the input features of each sample to an output class from the training data and build an ML classification model. The set of six input features are:
    - 1- receiving data rate in Kbytes/sec.
    - 2- transmitting data rate in Kbytes/sec.
    - 3- total received Kbytes.
    - 4- total transmitted Kbytes.
    - 5- total number of received packets.
    - 6- total number of transmitted packets.

These features were used as contextual inputs for training ML classifiers of output classes:

  - 1- high.
  - 2- varied.
  - 3- low.
  - 4- buffering.

- Evaluate the performance of ML classifiers using 10-fold cross-validation. Based on the result of the analysis, determine the more suitable ML classifier for classifying smartphone applications' network traffic reflecting varied types of network behaviour and interaction. Then, assess the generalisation capacity of the selected classification models on unseen testing data of applications that were not included in training data. Along with evaluation metrics, provide a confusion matrix to enable a detailed breakdown of the predictions, including the distribution of correct and incorrect predictions made by the classification models.
- Devise power saving modes based on the classified output traffic of the captured samples from a varied range of smartphone applications' network traffic.
- Evaluate the performance of the proposed power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, using varied sets of energy parameters. These energy parameters are:

  - 1- txPower: the power consumption during packet transmission.
  - 2- rxPower: the power consumption during packet reception.
  - 3- idlePower: the power consumption when a WNIC is awake and not transmitting or receiving packets.

- 4- transitionPower: the power consumption when a WNIC transits from the sleep to idle state and vice versa.
- 5- transitionTime: The amount of time required when a WNIC transits from sleep to idle state and vice versa.
- 6- sleepPower: The power consumption when a WNIC is in sleep state.

1.3 Contributions

In this thesis we proposed a novel concept of Context-Aware Listen Interval (CALI), in which the wireless network interface, with the aid of an ML classification model, sleeps and awakes based on the level of network activity of each application. This is further divided into the following more specific contributions:

- **Context-Aware Network Traffic Classification**

  We proposed a new ML based approach to classify the network traffic of wireless devices in WLANs. Smartphone applications' network traffic reflecting a diverse array of network behaviour and interaction were used as contextual inputs for training ML classifiers of output traffic. We employed five commonly used ML classifiers to classify the network traffic of a varied range of smartphone applications, firstly using 10-fold cross-validation for the initial classification, followed by extensive experimentation to assess the generalisation capacity of the selected classifiers on unseen testing data. The experimental results further validated the practical application of the selected ML classifiers, where the classification models have demonstrated strong generalisation capabilities and indicated that ML classifiers can be usefully applied for classifying the network traffic of smartphone applications based on different levels of behaviour and interaction.

- **CALI Power Saving Modes**

To optimise the sleep and awake cycles of the WNIC in accordance with the smartphone applications' network activity, we have developed four CALI power saving modes. These power saving modes enable additional power saving opportunities and have been devised based on the classified output traffic of the captured samples from a varied range of smartphone applications' network traffic. Hence, the ML classification model classifies the new unseen samples into one of the classes, and the WNIC will be adjusted to operate into one of CALI power saving modes. The experimental results have demonstrated that CALI power saving modes consume up to 75% less power when compared to the currently deployed power saving mechanism on the latest generation of smartphones, and up to 14% less energy when compared to Pyles' et al. SAPSM power saving approach, which also employs an ML classifier.

- **Dataset**

We have constructed a real-world dataset based on the network traffic of nine selected smartphone applications depicting different types of network behaviour and interactions; including, two VoIP applications, two applications of video calls, two applications of intermittent network interaction, two applications of very low network interaction, and finally one application representing applications with buffer streaming. This has resulted in the construction of a dataset, named Dataset 1, consisting of 1350 instances, with 150 instances per application and 6 features per instance. These features are statistical-based and unique for specific types of applications. Additionally, inspection into the packet content is not required to extract these features, hence statistical features have low computational overhead and are applicable for both encrypted and unencrypted traffic. Moreover, these features reflect the applications'

network interactivity better than non-network features like touch screen rate, as regularly touching the screen, does not always mean that network traffic is occurring. For instance, video games are highly interactive in terms of user and screen, but practically non-interactive in terms of network interaction. Furthermore, four output classes were assigned to cater for the network traffic of these applications. Thereby out of the nine chosen applications, the first output class was assigned to the four applications that represent real-time applications with high and constant levels of network interaction. The reason for having four applications for this output class is to ensure more variation in the range of network traffic included in the training data by having two VoIP applications and two video-calling applications. For the remaining three types of network traffic, the second output class was assigned to the two applications that represent network traffic with intermittent levels of interaction, while the third output class was assigned to the two applications that represent the least levels of network interaction. Finally, the fourth output class was assigned to one application that represents the network traffic of audio streaming applications. In addition, further datasets were constructed from Dataset 1 by the application of different feature selection algorithms. Dataset 2CBFS is based on a consistency feature selection algorithm and Dataset 3IGFS is based on an information gain feature selection algorithm.

1.4 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 - Background and Related Work: this chapter reviews state-of-the-art power saving protocols in IEEE 802.11 Family including their comparative drawbacks. This is followed by a critical review of power saving approaches proposed in the scientific literature. This chapter also discusses the

performance evaluation methods used in communication networks, with more attention given to the chosen method.

Chapter 3 - Network Traffic Classification: this chapter reviews the existing techniques used for network traffic classification, including ML methods. It then introduces and explains the essential steps required to perform an ML based traffic classification. This chapter also presents an ML classifier taxonomy, it then describes which ML classifier goes with which kind of data. This chapter also justifies the selection of the chosen ML classifiers. It then proceeds to describe the ML classifiers employed in this research, followed by presenting a comparison of ML classifiers. This chapter also provides an overview of other DL methods and illustrates how they differ from the ML ones. This is followed by an overview of ML classifiers. Lastly, it presents a review of proposed ML methods in the scientific literature.

Chapter 4 - Optimising WLANs Power Saving: this chapter discusses our novel Context-Aware Listen Interval framework for optimising WLANs power saving. It describes how different levels of traffic behaviour and interaction in the background of smartphone applications are used as contextual inputs for training ML classifiers of output traffic constructing an ML classification model. This chapter also justifies the selection of the chosen applications and the assignment of output modes. It also explains how the CALI power saving modes were used to optimise the sleep and awake cycles of the WNIC in accordance with the smartphone applications' network activity. This chapter also presents the process of data extraction and preparation used in this research to construct the dataset. It then proceeds to describe the experimental settings employed in this chapter for traffic classification, including the description of parameter settings for the selected ML classification models. Lastly, it evaluates the performance of ML classifiers using 10-fold cross-validation before and after the application of feature selection methods.

Chapter 5 - Experimentation: Analyses and Discussions: this chapter conducts extensive experimentation to determine whether the selected classification models generalise well on unseen testing data of applications that were not included in training data. This chapter also provides an in-depth analysis of the network traffic for the selected applications used in training and testing. To assess the generalisation capacity of the selected classification model, four main experiments are conducted in this chapter. For each experiment, it describes the experimental setup, followed by presenting the results and discussing the outcomes. This chapter also provides detailed conclusions based on conducted experiments. Lastly, it explores the feasibility of manually crafting rules to hand-classify the training data. Where an attempt to hand-classify the training data is made, followed by a discussion and comparison of the outcomes with the classification models constructed using ML classifiers.

Chapter 6 - Hyperparameter Optimisation: this chapter conducts the hyperparameter optimisation process using both manual and automated tuning methods to identify the optimal settings that result in a better-performing classification model. In this chapter, various hyperparameter settings were explored by performing 10-fold cross-validation firstly on the training data of experiment three consisting of 185 samples. Followed by evaluating the performance of the constructed classification models using the obtained optimal sets of hyperparameter values on the testing data of the same experiment. This chapter further assesses the performance of the classification models by repeating the previous four experiments conducted in chapter 5, using the optimal sets of hyperparameter values that were obtained through the optimisation process. Since the experimental results particularly of the repeated experiments one and four showed that using the optimised hyperparameters for a particular training data may not always lead to an improved model performance when there are changes in the overall

distribution of new training data, and the default hyperparameter settings in some cases perform comparably or better than the optimised hyperparameters. Thus, this chapter conducts further hyperparameter tuning, where the optimal sets of hyperparameter values were determined for classification models of the first and fourth experiments and the experimental results confirmed that better results can be obtained by conducting a hyperparameter optimisation process independently for each training data.

Chapter 7 - Performance Evaluation of CALI Power Saving Modes: this chapter evaluates the effect of adjusting the WNIC on energy consumption after the accomplishment of the classification process using an ML classification model. It describes experimental setup employed in the creation of the corresponding traffic scenarios of CALI power saving modes. It then assesses the performance of CALI power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, using varied sets of energy parameters. This is followed by assessing the performance of CALI against the value variations of energy parameters.

Chapter 8 - Conclusion and Future Work: this chapter summarises the thesis, reviews the objectives and discusses how they were addressed. Finally, this chapter highlights the limitations and outlines possible future research directions.

2

BACKGROUND AND RELATED WORK

2.1 Introduction

This chapter explains and critically reviews the state-of-the-art power-saving protocols in the IEEE 802.11 Family. This is followed by a critical review of power-saving approaches proposed in the scientific literature. This chapter also discusses the performance evaluation methodologies in communication networks, it then describes the employed method in this thesis to evaluate the performance of CALI power saving modes.

2.2 Related Work

This section reviews the deployed power saving protocols in WLANs, in particular SPSM and APSM including their comparative drawbacks, and further developments of power saving protocols in the IEEE 802.11 Family. This is followed by a critical review of power saving approaches proposed in the scientific literature.

2.2.1 PSMs for IEEE 802.11 Family

2.2.1.1 Static PSM

In the WLAN Infrastructure Basic Service Set (IBSS), the 802.11 standard defines SPSM to reduce the amount of energy consumed by the WNIC when the wireless devices are connected to an AP. The WNIC of a wireless device in SPSM operates in two modes: awake mode and sleep mode. In the awake

mode, the radio transceiver of a wireless device is on, fully powered and ready to receive and transmit consuming a significant amount of power. While in sleep mode, the radio transceiver of a wireless device is not fully powered, meaning that the wireless device cannot receive or transmit in order to conserve power [1].

In SPSM, the AP announces the presence of any buffered packets intended to a wireless device via a Traffic Indication Map (TIM) in a beacon frame. Thus, the wireless device stays in sleep mode and periodically wakes up during its listening interval (multiples of the beacon interval) to listen to the TIM in the beacon frame. If the TIM does not indicate packets for the wireless device at AP, the wireless device immediately goes back into sleep mode to save power.

In the case a TIM indicates the existence of buffered packets at AP, the wireless device remains awake and generates the Power Save Poll (PS-Poll) frames to retrieve the buffered packets from the AP. Upon receiving the PS-Poll frames, the AP transmits the buffered packets to the wireless device, one packet at a time and receives its corresponding Acknowledgment (Ack) until all buffered packets are received successfully and the AP finally indicating the existence of no more packets by setting the value of the More Data field to zero [2, 3]. Figure 2.1 illustrates the operation of SPSM in IBSS.

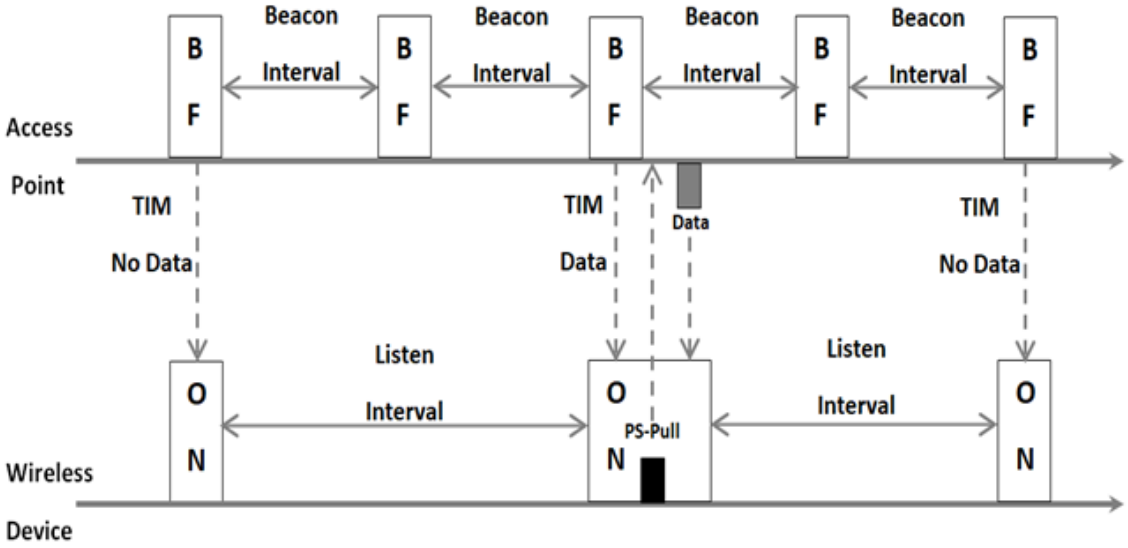


Figure 2. 1: Static PSM

So, the Wireless device wakes up and turning on its receiver in the listen interval to listen for TIM, TIM which is sent by AP through beacon frame indicates that, the AP has no buffered packet for the wireless device, so the wireless device immediately goes back into sleep mode, and skips the following beacon, because the listen interval is a multiple of beacon interval. Now again the wireless device wakes up and listens to the third beacon interval, this time the TIM indicates the presence of buffered packets for the wireless device in the AP.

Now, the wireless device sends a PS-Poll frame to AP requesting its buffered packets, these PS-Poll frames are sent by the wireless device according to Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). AP receives the Poll-Frame from the wireless device and then starts transmitting the buffered packets one by one, (transmits one packet and receives its corresponding ack frame from the wireless device and so on).

So, the wireless device keeps sending the PS-Poll frames till the value of more data field in the data frame set to zero, which means there are no more packets buffered in AP for the wireless device.

The SPSM conserves energy by allowing a wireless device to sleep and wake up periodically. Nevertheless, SPSM suffers from latency issues for the following two reasons: firstly, when a wireless device generates the PS-Poll frames in order to retrieve the buffered packets one at a time from AP [4, 5]. Secondly, when a delay of 100-300ms is introduced when the WNIC is off during the beacon intervals and there are incoming packets for a wireless device buffered at AP [6]. These issues impact on the performance of both, real-time applications such as VoIP and interactive applications such as web browsers.

2.2.1.2 Adaptive PSM

APSM has been deployed within the most recent generation of mobile devices to overcome the delay of the WNIC being off during the beacon interval and the delay caused by the PS-Poll frames. In APSM, a wireless device adaptively switches between sleep and awake mode depending on the network traffic [7]. In APSM, by default, a wireless device remains in SPSM [8]. To switch from SPSM mode to the awake mode, the wireless device notifies the AP by sending a null data frame with the power management bit set to zero. When the AP receives the null frame, it stops buffering packets for the wireless device.

To switch back into SPSM mode, the wireless device sends a null data frame with the power management bit set to one, so that the AP resumes buffering packets for the wireless device [5, 9].

APSM operates based on a threshold, i.e., ingress and egress packets between a timer start and expiry are counted. If the counted packets exceed the threshold, the WNIC switches to the awake mode. On the other hand, if the counted packets are below the threshold, the WNIC remains in SPSM mode [10]. Figure 2.2 shows the threshold mechanism of APSM.

Latency related issues found in SPSM are eliminated in APSM. However, the WNIC of a wireless device does not take into consideration the type of network traffic, whether this type of network traffic is important or not, instead it switches from sleep to awake mode based on network activity thresholds.

This may lead to the WNIC being switched into awake mode unnecessarily, receiving low priority traffic consuming energy which could be better used for more important traffic [10, 5]. Moreover, the WNIC remains in awake mode for an idle timeout period before being fully switched back to SPSM [11].

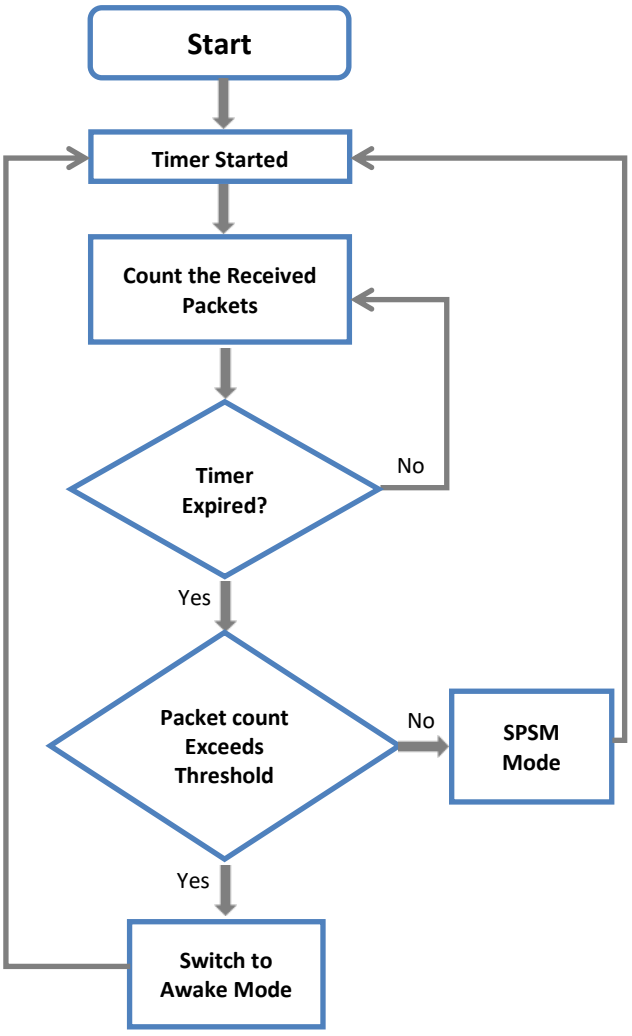


Figure 2. 2: Threshold mechanism of APSM

2.2.1.3 Other Power Saving Protocols in IEEE 802.11 Family

Automatic Power Save Delivery (APSD) has been presented in the requirements of IEEE 802.11e. It includes Quality Access Point (QAP) which automatically delivers downlink frames to power-saving devices, thus avoiding the regular need for polls for each frame. APSD identifies two delivery schemes, Unscheduled U-APSD and Scheduled S-APSD [12].

An upgraded version of APSD called Power Save Multi Poll (PSMP) mode, was introduced in the 802.11n standard, this protocol has two modes: Unscheduled Power Save Multi Poll (U-PSMP) mode and Scheduled Power Save Multi Poll (S-PSMP). The issue with PSMP is the higher occurrence probability of buffer overflow which may cause packet loss. This is due to the fact that frames are buffered for a longer time at the AP.

In S-PSMP mode, AP delivers schedules to wireless devices that provide information on precise time intervals where the frames will be sent. Therefore, wireless devices sleep for the majority of their time and then wake up at the scheduled time only [13].

In U-PSMP mode, the AP is informed by a wireless device that all designated frames must be buffered and sent only when AP receives a frame from that wireless device. Therefore, a frame serves as a trigger that causes the AP to send frames to a wireless device instantly [14].

Another antenna-compatible technique named Spatial Multiplexing Power Save (SMPS) was also adopted in 802.11n standard. This technique enables the wireless device to power down all but one of the reception chains. However, the issue with this mode is the decreased number of receiving chains which contribute to lower overall communication link performance [15].

Target Wake Time (TWT) mechanism was adopted under the IEEE 802.11ah standard in order to avoid the long-time listening for Beacon [16]. An AP and involved station will arrange the exploitation of TWT features and schedule a particular time to access the media to a particular station. The AP and the station share details such as the operation time estimation. The AP will then monitor the superposition and competition for the required media access by the station to prevent collisions and contention between different stations. The sleep mode could be entered prior to TWT by using the TWT to decrease power consumption. The TWT in 802.11ax is divided into two sorts: broadcast TWT and Individual TWT. The individual TWTs require individual TWT agreements between the TWT scheduling AP and scheduled stations, whereas it is not required in broadcast TWT [17].

2.2.2 Sleep Optimisation (Extending Sleep Period)

Li et al. [18] proposed Dynamic Listen Interval (DLI) to reduce the energy consumption caused by unnecessary wakeups. In this scheme, the listen interval of a wireless device is incremented by 1 each time a wireless device wakes up during its listen interval and finds the presence of no packets buffered at AP. Moreover, a wireless device reverts its listen interval to 1 when it finds the presence of buffered packets at the AP. The proposed scheme conserves power in comparison with SPSM by adjusting longer listen intervals, but an additional delay will be added if packets of interactive applications are buffered at an AP during the increased listening interval.

Attempting to eliminate the issues related to APSM, Pyles et al. [10] proposed SAPSM, which is based on categorising smartphone applications as either low or high priority apps using an ML classifier. Consequently, the traffic of applications, which have been tagged as high priority, switches the WNIC into awake mode. Conversely, network traffic of low priority applications keeps the WNIC in SPSM conserving energy. To train the ML classifier and set

applications' priority, a study was conducted. In this study, participants interacted with a range of applications that have diverse levels of network interactions. Initially, all applications are configured with SPSM, and based on the participants' experience with the selected application, the priority of each application was determined. The priority is set to high if the observed delay by a participant is unacceptable. In contrast, it is set to low if the observed delay has not impacted the participants' experience. The Support Vector Machine (SVM) classification model that was used in the study has achieved an accuracy of 88.1%.

However, no further priority levels or modes have been proposed in this work to cater for applications with different patterns of network activity: including the least levels of network interactivity, that receive network updates after longer periods of time, secondly, applications with intermittent network interactions, and finally, for applications with buffering capabilities. Instead, SAPSM operates the WNIC in SPSM for all low priority applications.

Li et al. [19] introduced a similar approach to SAPSM, which is also based on prioritising smartphone applications into low and high priorities. Authors of this approach conducted measurements of smartphone applications' usage. Based on these measurement results, two features that reflect network interactivity: the receiving rate and the screen touch rate were extracted. Finally, based on these two features, a prioritisation scheme that classifies applications' network traffic into low or high priorities was presented. For high priority applications the network traffic will be operating in the awake mode, and for low priority applications the network traffic will remain operating in SPSM. The proposed scheme in [19] was only evaluated against a user study. Moreover, no further priority or mode was considered for applications that are capable to operate with extended periods of WNIC listening intervals.

Kwon and Cho [20] proposed a simple priority scheme inter-user Quality of Service (QoS). In this proposed scheme, the priority of each wireless device is already defined. So, the AP retrieves the user profile information from the Authentication, Authorization and Accounting (AAA) server upon the registration of the wireless device. Thus, after determining the priority for each wireless device, the wireless device with high priority is allowed to send the PS-Poll frames to retrieve its buffered packets immediately and earlier than the wireless device which has been assigned as low priority. Therefore, a high priority wireless device goes into sleeping mode for the rest of the beacon interval. Moreover, wireless devices set with the same priority levels are contended to access channel according to the Distributed Coordination Function (DCF) scheme. However, the proposed scheme is only beneficial and in the favour of high priority wireless devices, as high priority wireless devices can fetch their buffered packets faster with minimal delay. But causing delay and energy consumption for low priority wireless devices, as they keep sensing the channel till other wireless devices with higher priorities finish capturing their buffered packets from the AP.

Authors of [21] propose Catnap, a system which decreases wireless devices' power consumption by enabling them to sleep while transferring data. It utilises wired and wireless bandwidth discrepancies to optimise wireless device's power saving. By integrating small gaps into significant sleep periods between packets, Catnap uses high bandwidth-wireless interfaces that have far greater bandwidth than the bandwidth available over the Internet to allow the device including its WNIC to doze off. Catnap's core elements are an independent application proxy that separates wireless and cabled segments; also, a scheduler that runs on Application Data Units (ADUs) to optimise mobile device sleep duration without affecting the average time of ADU transfer. Catnap is intended for data-oriented applications like browsing and

transfer of files so that individual packets are delayed but the overall times of transfer are not increased.

In [22] researchers presented an approach called Snooze, which is an 802.11n energy management strategy that incorporates a micro sleeping method with an antenna configuration management technique. Moreover, micro sleeping allows the WNIC to sleep at low power levels during few milliseconds and the antenna configuration management adaptively changes the amount of powered RF chains. In order to enhance the energy efficiency, Snooze adjusts user sleep and antenna configurations for the entered/out traffic of wireless internet packets; this adjustment is performed by a traffic shaping which provides sleep possibilities while minimising latencies, and also taking the inter-reliance between the microsleep and antenna configurations into consideration.

Pyles et al. [23], propose a silence prediction approach called SiFi. Applications such as VoIP do not operate well in PSM mode since the power footprint is fairly high in real time. SiFi provides a technical approach for this kind of application. SiFi checks audio streams of smartphone calls and monitors from the start till the stop of silence intervals. These parameters are contained in prediction models, with the help of these historical records, future silence periods are predicted and used to set the WNIC into sleep mode.

In [24], authors propose a solution called Micro-Power Management (µPM) which is inspired by the incompatibility between the high-performance 802.11 standards and the moderate data rate specifications of a wide range of common network applications. The µPM allows an 802.11 wireless device to join saving modes such as the one between MAC frames. To handle data loss, µPM employs the re-transmission technique in 802.11 and manages frame delays, with limited cooperation from the AP.

The emphasis of this research is on reducing power in the brief idle periods, which is not handled by 802.11 PSM. Such idle periods are abundant due to the difference between the high data rate enabled by current 802.11 standards and the moderate data rates requested by several common applications. In fact, it is also due to the wired link limitations such as DSL.

The study of [25] propose a Centralized-PSM (C-PSM) an AP's Centric PSM for 802.11 networks. C-PSM optimises the overall power consumption for wireless devices by allowing the AP to select the optimum PSM parameters including, the listen and beacon intervals according to the client's traffic pattern. These periods are adjusted to decrease power consumption due to excessive wake ups and contentions of the channel. As the power loss in the contention can be very costly since all the involved wireless devices cannot turn to sleep mode during the time of contention. Furthermore, the AP provides wireless devices with optimum congestion time windows such that the less frequently active device can retransmit faster. Moreover, C-PSM offers the first wake up schedule to help improve power efficiency by minimising simultaneous wake ups for wireless devices.

Tan et al. [26] propose a PSM-throttling, in which a wireless device identifies a bandwidth throttling link by detecting the TCP flow throughput. As a result of the throttling, the client can anticipate when a packet will arrive and switch on/off the WNIC by reshaping TCP traffic into intermittent bursts of the same average rate as the server transmission.

In [156] Jiang et al. proposed QoS-aware architecture to reduce the energy consumption in Wi-Fi networks for wireless devices steaming Danmu videos. Danmu also known as bullet comments or barrage videos, is a special type of interactive video streaming that shows users' comments on top of the videos during the playback. As the network transmissions of video related packets from the streaming server and barrage related packets from and to the mobile

user device are not coordinated for Danmu videos. Authors of [156] deployed a client proxy on a mobile user device and an edge proxy on a home Wi-Fi router, both proxies are used to coordinate the transmissions of video data and barrages.

To improve the energy efficiency of APs in WLANs, the authors of [159] proposed a reinforcement learning-based solution. The proposed solution considers the network conditions such as queue length, and channel gains to the user in order to control AP's transmit power and determines whether an AP should use single or bounded channels.

Venkateswaran et al. [160] propose a power saving optimisation algorithm for low powered IoT devices operating under Wi-Fi networks. In this work, authors have only considered a specific scenario of sparse periodic uplink traffic. This is the scenario where an IoT sensing device periodically reports on measurements such as soil moisture level, temperate and so on, or sends keep-alive packets to a remote server over a Wi-Fi-based AP. Authors firstly introduced five potential energy optimisation strategies including SPSM, based on simulations results, they have developed the power saving optimisation algorithm.

Seth et al. [163] proposed an EAPS, 802.11 AP based solution to reduce the power consumption of IoT devices by minimising the duration of the idle listening. In this work, upon the reception of uplink packets from an IoT device, the AP computes the estimated delay that occurs till the reception of downlink packets using an ML-based model. Once the estimated delay is computed, the AP informs the device of the scheduled time to wake up in order to receive the downlink packets.

In relation to harvesting energy from WLANs, wherein wireless devices can be charged via radio frequency signals emitted from an AP. The work in [161]

presented a transmit power allocation policy for a solar-powered AP. The proposed policy allows AP to control the transmit power in order to deliver the required data to nonenergy harvesting wireless devices such as laptops, iPads and so on. And simultaneously ensures the IoT sensing devices with energy harvesting capability receive sufficient energy. Finally, to derive the policy, the authors employ ML approaches to determine the optimal transmit power that satisfies the requirements of both types of wireless devices based on AP's current and historical battery status.

To examine whether the application of different schedulers for TWT scheduling improve throughput and energy efficiency. Yang et al. [164] presented max-rate and proportional fairness schedulers to enhance the throughput and energy efficiency of TWT capable wireless devices operating under 802.11 ax. The findings of this research show that applying different schedulers for TWT will improve the energy and enhance the throughput.

To improve the throughput and energy efficiency of wireless devices in a coexistence area of Wi-Fi and cellular networks. Zhang et al. [165] propose TAUD scheme, the proposed scheme splits the wireless devices in the coverage area of dual networks into two groups: WLAN-based group, and cellular-based group. Thus, wireless devices in the WLAN group only connect to the Wi-Fi system whereas wireless devices in cellular group connect to a cellular network.

802.11ah was introduced to accommodate the dense IoT networks. In this multi-rate network, different IoT devices have different data rate requirements. And IoT devices access the channel using group based Restricted Access Window (RAW) mechanism. As the standard does not specify any scheme for group forming, by default a uniform grouping scheme is employed, which forms random groups of an equal size, resulting in a performance anomaly. To resolve this issue, and optimise energy efficiency,

the Authors of [166] proposed RAW-RA scheme, in this scheme the IoT devices are grouped based on data rates and the RAW slots are assigned to the groups proportional to their data rates.

2.2.3 Handling Traffic Contention

Rozner et al. [5] introduced a Network Assistant Power Management solution (NAPman). The authors conducted a variety of experiments to show that current implementations of PSM strategies in wireless devices and APs are not efficient due to competing background traffic which increases the energy consumption of a wireless device and decreases the network capacity due to unnecessary retransmissions. To mitigate these issues, NAPman employs virtualisation and an energy-aware scheduling algorithm for AP based on the First Come First Serve (FCFS) policy that applies only to packets of wireless devices that are awake at a given time. By leveraging AP virtualisation, contention among wireless devices is mitigated, as several virtual APs from one physical AP are created. Each wireless device is connected to its own dedicated copy of a virtual AP. As NAPman relies on virtualisation, one physical AP can only support a limited number of virtual APs. This causes disruption when the number of assigned wireless devices to virtual AP exceeds the threshold limit.

In [27] He and Yuan propose a time division multiple access approach based on MAC protocol, called scheduled PSM. In this approach, the beacon interval is divided into an equal number of slices by an AP. The slices can be assigned to a single wireless device or multiple wireless devices. The TIM was restructured to hold slice assignment information. Scheduled PSM eliminates channel contention, as each wireless device wakes up on its designated time slot to retrieve the buffered data from the AP, and sleeps during its non-allocated time slots to save power. This approach conserves energy as the channel is contention free, but time slots will be wasted if a wireless device

does not wake up at its designated time slot. Also, this approach suffers from additional delay: data frames arriving at the current beacon interval will only be scheduled for transmission to a wireless device in the next beacon interval. Finally, all the time slots are identical in size, which may not be appropriate for small frames or light traffic.

Opportunistic Power Saving Mode (OPSM) is proposed in [28]. The application of OPSM is limited to a specific scenario: wireless devices are engaged in web browsing to download short files with a short duration of inactivity or think time in between downloads. The authors of [28] observed that the throughput share of an individual wireless device decreases in SPSM when multiple wireless devices are associated with a single AP and download files simultaneously. Therefore, to gain the maximum throughput and reduce energy consumption, only one wireless device is permitted to download a file at a time in OPSM. During this time other wireless devices remain in sleep mode. One additional bit has been added to the beacon header indicating whether the AP is currently serving another wireless device. To avoid a number of wireless devices from initiating a file download simultaneously on completion of the service of the current wireless device, wireless devices wait for a random period of time before initiating their file download.

In [29] Omori et al. present a power saving approach that utilises Network Allocation Vector (NAV) periods set by the Request to Send (RTS) and Clear to Send (CTS) handshake mechanism. The proposed approach allows other wireless devices to sleep when they overhear the CTS or RTS during the NAV duration. Moreover, the NAV duration is extended which allows multiple bidirectional burst transmission between a device and an AP. In their previous work [30] the authors of this approach utilised NAV duration by allowing the burst transmission in an unidirectional manner for incoming packets from AP only.

Authors of [31] proposed an energy efficient technique called SleepWell that avoids wireless network contentions. The APs control the client's sleeping window so that various APs are awake/sleep in time windows which are not overlapping. The approach is similar to the general insight of late arriving at the office and late leaving, thus eliminating hours with congestion. As APs are always on, wireless continuous traffic from neighbouring APs is monitored. Because SPSM periodically causes traffic bursts, each AP monitors the regularity of the other APs, and re-plan its own period dynamically to marginally overlap with the others. Lesser overlap eliminates competitiveness so that any client can download their own packets continuously and sleep while other transmissions are established on the channel.

Authors of [32], propose Harmonious Power Saving Mechanism (HPSM) which addresses the situation of many PSM clients linked with a single AP. The core principle of HPSM in handling the traffic contention, is to utilise the underlying sociological principle [33]. The connecting resource and the battery life of wireless devices are defined as public and private resources, respectively. When a PSM client uses large amounts of public resources, it is considered wealthy and vice versa for poor devices. Similar to real life societies, poor citizens cannot obtain more costly public facilities such as higher education, but they pay significantly less tax. Correspondingly, devices utilising a significant portion of the public resource should pay for the service more in a network consisting of one AP and several PSM clients, but those that consume a limited portion of the resource should have less expensive service cost. HPSM deliberately places priority on the transfer of data for poor or weak over wealthy devices. Thus, as the data transfer is completed early, the weak wireless devices will go back to sleep mode for greater power savings. In the meantime, since poor wireless interfaces only use the communication channel for very short times, the taxes charged by these rich wireless devices on latency and power are small.

Authors of [34] found that, the standard FCFS mechanism is unideal for sleep since it keeps the PSM wireless devices awake needlessly. A downlinking traffic scheduler called SOFA on the AP is proposed. It saves power by encouraging its PSM devices to sleep more while the service of other PSM devices. If a device's buffered packet exists in the AP and it chooses to retrieve it, it must be kept awake till the last packet planned for it is provided during the time cycle of the beacon. As the AP delivers packets to other wireless devices a substantial amount of power consumption occurs before the last packet of the wireless device is transmitted to its destination. SOFA decreases that energy consumption and enhances all wireless devices' overall sleep time. This is done by calculating the quota of all wireless devices and forwarding packets based upon the calculated quota.

To minimise the number of contention's wireless devices, researchers in [35] proposed Load-Aware Wakeup Scheduling (LAWS), a technique which organises wakeup schedules for sleeping wireless devices such that the amount of wakeup devices are balanced for each beacon interval. To minimise both collision likelihood and energy usage, the AP announces a subset of PSM clients in the beacon frame which is exploited by the client to decide their polling sequence. Three methods for access schedules were proposed to prevent contention. First, the multiple wakeups single access, where only a single wakeup device is planned for access to the buffered data. Second, multiple wakeups multiple access with smallest Association ID (AID), where the access's planning is based on the smallest AID. And finally, multiple wakeups multiple access with the smallest queue length first scheme, where the shortest queue length is first in a beacon interval.

2.2.4 PHY-Assisted Power Saving

In [36] a technique called DozyAP is proposed to enhance the energy efficiency of Wi-Fi tethering. Based on a conducted analysis of traditional applications,

authors of [36] have identified various occasions that the tethering phone could sleep to conserve energy. DozyAP's main concept is to place the Wi-fi tethering based phone which acts as mobile AP's Wi-Fi interface in the sleep mode to reduce power consumption. After analysing the traffic pattern of different online applications. Authors noted that the Wi-Fi network is inactive for a significant duration of the overall application time so that the AP could sleep for this idle time. In addition, it is well known that a cellular link is usually slower than a Wi-Fi link. Therefore, when waiting for data transmission via the cellular network, the Wi-Fi interface of a mobile AP can sleep. All these show that mobile AP energy demand can be reduced in several ways.

Similarly, GreenAP has recently been introduced in [157] to solve the issue of higher power consumption of mobile AP. Two strategies were considered in the design of GreenAP: first, delaying the transmission of AP's sleep indication frames in order to minimise the traffic delay when AP encounters incoming packets from a wireless device. Second, is the selection of an appropriate sleep time for the AP based on the mean traffic statistics.

An energy-efficient AP selection approach for IoT nodes in hybrid Wi-Fi and Li-Fi I networks is presented in [158]. In the proposed solution, the IoT client selects either a Wi-Fi AP or a Li-Fi AP based on the satisfied QoS requirement of throughput. In the case of both APs provide the same satisfaction of QoS constraint. An AP which provides a better energy efficiency is chosen. Whereas the node remains in Li-Fi based AP when none of the APs satisfies the QoS constraint.

The authors of [162] have proposed a computation offloading technique to reduce the energy consumption of wearable devices. In this work, when a task is given in a wearable device such as a smartwatch, the wearable device firstly decides whether to execute it locally or offload it to a smartphone. The

offloading decision is made by comparing the cost of operating the task locally and the cost of energy consumption when offloaded to a smartphone along with the cost of the Bluetooth communication between a wearable device and a smartphone. Based on this proposition, a wearable device offloads the computation task to the smartphone.

Now, when a smartphone receives the offloading request, it decides whether to execute the task locally or offload it to a cloud server based on the battery's remaining energy and the cost associated if the task is handled locally in the smartphone. In case the total cost is high, then the smartphone offloads the computation task to the cloud via a Wi-Fi connection.

Studies [6] and [37] explore conserving power by utilising multiple radios of wireless devices. Authors of [6] introduced Bluesaver, which employs Bluetooth and Wi-Fi combined at an AP and wireless device. The wireless device switches between Wi-Fi and Bluetooth radios. The wireless device receives and sends packets over Bluetooth when it is within range of the Bluetooth radio of the AP. When a higher data rate is required or a wireless device is out of range of the Bluetooth radio of the AP, it switches to Wi-Fi radio. However, this approach requires an additional Bluetooth adaptor at the AP.

Zhang and Li [37] developed a Wi-Fi ZigBee message delivery scheme, which delegates some of Wi-Fi operations to ZigBee radio. In this case, the Wi-Fi radio of a wireless device is turned off, and instead, low power ZigBee radio is utilised to discover the presence of Wi-Fi networks. It then listens to incoming beacon frames from the AP to detect the presence of any buffered packets intended to a wireless device. However, the developed scheme in [37] requires an external chipset on smartphones.

Similarly, in [38] authors have introduced a Blue-Fi, a system that detect the presence of Wi-Fi AP through a Bluetooth and cell tower information. This allows the wireless device to turn on the Wi-Fi interface only during the presence of Wi-Fi connection, thereby eliminating lengthy periods of idle time and reducing the number of scans for exploration substantially.

Vergara and Nadjm-Tehrani [39] proposed Watts2Share, an architecture that enhances power saving by consolidating the traffic in only one link. The proposed architecture combines multiple nodes for communication and use a single node's 3G interface for data transferring. The Wi-Fi radio is used as the secondary radio whereas the 3G channel is employed as a primary channel for communication.

Chung et al. [40] present C-SCAN, which utilises a low power personal wireless network interface that is embedded in the wireless device such as Bluetooth and ZigBee, to perform the unnecessary scanning of AP-free Wi-Fi channels to unload Wi-Fi scanning overhead. C-SCAN examines Bluetooth radio channel information and detects which Wi-Fi channels are in operation before the real Wi-Fi interface channel is scanned. The Wi-Fi scanning manager can search only on available Wi-Fi channels by removing channels that are determined to be null. Thereby delay reduction and energy efficiency improvement are achieved.

Other studies [41-43] focused on decreasing the radio's clock rate to conserve energy. SloMo [41] proposed a transceiver that enables a wireless device to operate at a lower clock rate during transmitting and receiving. E-Mili [42] allows the WNIC to operate at a lower clock rate during idle listening and transits to the full clock rate during data transmission and reception. In [43] the authors proposed Sampless Wi-Fi, which enables the wireless device to recover under-sampled packets via multiple transmissions.

The authors of [167] developed a multi-mode transceiver combining a set of wireless connectivity protocols. The transceiver was designed for IoT wireless devices and supports Wi-Fi, Bluetooth classic and Bluetooth low energy wireless connectivity protocols. The reported measurement results in this study show that the developed combo chip achieves better or comparable performance in comparison with standalone chip architectures.

2.3 Evaluation Methodology

This section discusses the most commonly used methods in communication networks for evaluating the performance of a proposed system or framework, it then describes the employed method in this thesis to evaluate the performance of CALI power saving modes.

Typically, the efficiency of communication networks can be assessed using three main methods. These methods are real experimental, analytical modelling and simulation [49, 50].

The real experimental method requires the use of a testbed to evaluate the proposed approach. Typically, a testbed comprises a set of wireless components, which are designed to run the proposed approach. The benefit of using this method is that realistic circumstances are taken into consideration, which provides a better insight of the proposed approach before implementing it in the real world.

Nevertheless, it is costly and time consuming to build and maintain a testbed, particularly when assessing the efficiency of the proposed approach in larger networks. The required components, individuals, and coding cause higher costs for building and maintaining a testbed. Thus, teamwork will be necessary for this method.

On the other side, the mathematical description of the proposed approach, i.e., analytical modelling, is based on applied mathematical theories including stochastic process and queuing. Numerical techniques may then be added to the model to have more insight into the model.

Mathematical modelling fits into basic networks which are comparatively small. It may become more difficult to derive certain models for a complex network such as WLAN. As, this type of network involves taking into consideration several factors, certain assumptions in the model may simplify some factors. Simplified assumptions may therefore generate imprecise results which influence the validity of the system analysis [50].

Simulation tools may also be used to test the performance of the proposed approach. A network simulator is designed for simulating a computer network including the operation of the networking nodes and links. The simulation approach has benefits over the other methods of assessment, such as having scalability, repeatable outcomes, and simulating complex scenarios.

In comparison with analytical modelling, the simulation based assessment often requires fewer assumptions. It also enables the designer to characterise the analysed approach in more detail. Simulation is a cheaper alternative compared to the real experiment method. Also, the efficiency of the proposed approach under different loads and various network scenarios can easily be investigated.

Therefore, to evaluate the performance of CALI power saving modes, the simulation based evaluation method has been adopted in this thesis.

2.3.1 Simulation Tools

A varied range of network simulators are used for simulating wireless networks such as NS [51], OMNeT++ [52], Riverbed [53], and QualNet [54]. These simulators are briefly discussed in the following paragraphs.

- **NS-2**

Network Simulator 2 (NS-2) is a simulator of discrete network events. This includes models for simulating cable and wireless networking standards on various levels, including network, data link, and physical layers. NS-2 was employed extensively in scientific research. It utilises C++ as the language of programming. Apart from C++, the Object Extension Tool command language (OTcl) of MIT allows the utiliser to define the C++ code parameters. NS-2 was substituted by its NS-3 successor. NS-3 was also designed with C++ from scratch. It emphasises the resolution of current NS-2 simulator issues, but it does not support all the NS-2 models since it has a compatibility problem with NS-2. NS-3 utilises C++ or python scripts. These simulators are open source and accessible free of charge for research, development, and utilisation.

- **Riverbed**

Riverbed is a discrete event simulator previously called OPNET, Riverbed consists of C++ based protocols and technology suit. It supports wired and wireless networking protocols, such as IEEE 802.11 a, b, and g standards. Besides, it is available as a commercial simulator and is one of the most commonly used network simulators. Riverbed offers excellent interactive support as a commercial simulator, enabling an individual to construct network model objects from the application layer to the physical layer. The users can also interpret and simulate the effects of the simulation using the available graphical interfaces.

- **QualNet**

QualNet is a network simulator based on GloMoSim simulator for commercial use. The GloMoSim simulator, which was developed for MANETs, is also a discrete event simulator. QualNet is GloMoSim's industrial derivative, which is not supported any more by its developers. QualNet provides varied wired and wireless network technologies simulations. QualNet offers a good user experience by its graphical interface which enables users to build the components of their scenarios and to set their parameters. It moreover offers excellent methods for analysing simulation outcomes. QualNet utilises C/C++ to build novel models, but it is not commonly employed in research because it is commercial.

- **OMNeT++**

OMNeT++ is a C++ object-oriented, extensible, flexible simulation library and network simulator platform. It is not an individual simulator but offers simple tools that enable developers to design their own simulators. Frameworks are individual projects that complement particular environments. For instance, there are frameworks supporting wireless networks, modelling of the performance, internet protocols, Peer to Peer (P2P) overlays, etc. The open-source OMNeT++ platform is provided for academic use, free of charge. It provides comprehensive graphical interface support. Simple modules, which are written in C++, are assembled for components of greater size. A high-level language known as Network Description (NED) that operates in an identical way to OTcl in NS-2 is used to model the components.

2.3.2 NS-2 Extension

For the performance evaluation in this thesis, we have utilised the NS-2 simulator. To support the power management functions in WLAN, we used

the NS-2 extension proposed in [55], which has been applied in several studies including [32] and [56].

This NS-2 extension provides PSM mechanisms, such as the PS-Poll, AP buffer, and TIM. Furthermore, it includes an energy model which uses four energy parameters: txPower, rxPower, idlePower, and sleepPower.

The extension supports infrastructure mode where two wireless devices are connected to an AP based on PSM. The first wireless device sends data destined to wireless device 2 via AP.

Next model's operational behaviour is shown graphically as the model moves during the simulation. This is followed by the displaying the remaining power of the receiver device in both scenarios (with PSM enabled and disabled). We used the default simulation parameters shown in table 2.1, with a simulation duration of 600 seconds and initial energy of 100 J.

Table 2. 1: Simulation parameters

Simulator	NS-2.33
Routing Protocol	DumbAgent
MAC type	802.11
Antenna model	Omni Antenna
Number of devices	2 Wireless devices (sender and receiver) and 1 AP
Simulation time	600 S
Initial Energy	100 J
Packet Size	512B
Data rate	256 KB
txPower	0.660 W
rxPower	0.395 W
idlePower	0.035 W
sleepPower	0.001 W

Figure 2.3 shows the sender and receiver wireless devices surrounded with a blue hexagon which indicates that the wireless devices are in sleeping mode. The AP is denoted as 0. Figure 2.4, shows that, the wireless device 1 wakes up to send data to wireless device 2 through AP.

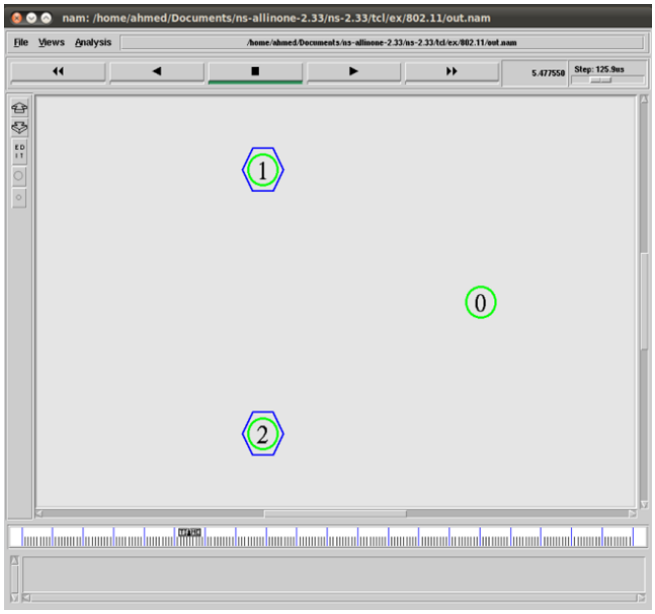


Figure 2. 3: PSM operational behaviour 1

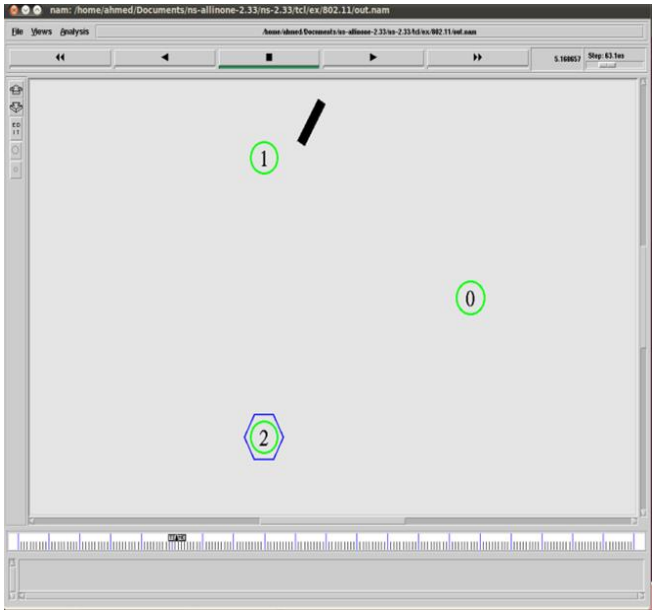


Figure 2. 4: PSM operational behaviour 2

Figure 2.5 shows that, both wireless devices are awake, and the wireless device 2 sends the PS-Poll frames to retrieve the buffered packets from the AP. The AP transmits the buffered packets to wireless device 2, one packet at a time and receives its corresponding Ack. Figure 2.6 shows, the wireless device 2

immediately switches into sleep mode after receiving all its buffered packets from the AP.

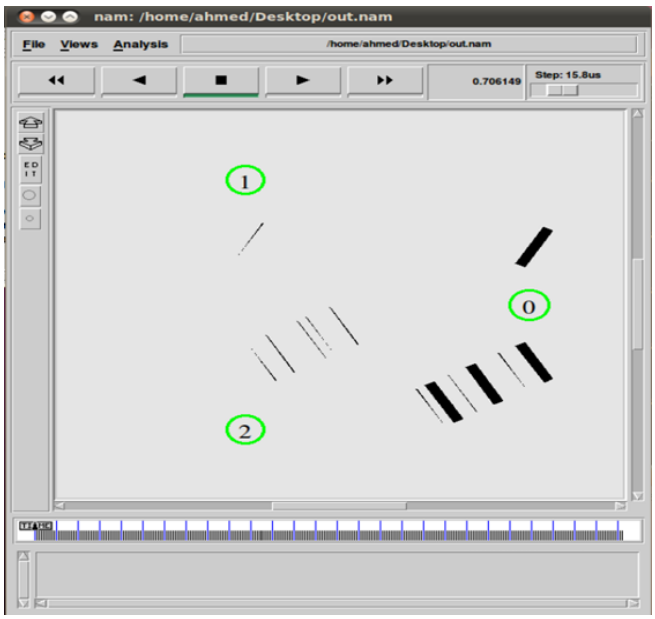


Figure 2. 5: PSM operational behaviour 3

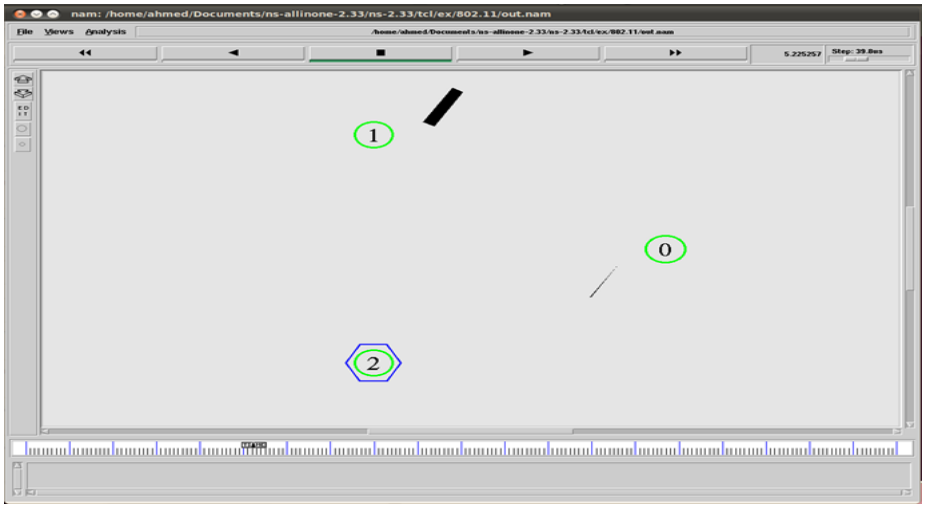


Figure 2. 6: PSM operational behaviour 4

The comparison between PSM enabled and disabled is shown in figure 2.7. The remaining energy is 71.69 out of 100 when the wireless device 2 (the receiver device) operates on PSM. While the remaining energy of the same wireless device falls to 50.48 in case the PSM is disabled.

Figure 2. 7: PSM vs. PSM Disabled

2.4 Summary

This chapter presented and critically reviewed the static and adaptive power saving mechanisms deployed in WLANs, including other power saving protocols in the IEEE 802.11 Family. It also has critically reviewed the power saving approaches proposed in the scientific literature. This chapter also has investigated the performance evaluation methods adopted in communication networks. This is followed by demonstrating the employed method in this thesis to evaluate the performance of CALI power saving modes.

3

NETWORK TRAFFIC CLASSIFICATION

3.1 Introduction

This chapter starts by explaining the concept of network traffic classification (section 3.2), it then discusses the existing classification techniques including ML methods used for classifying the network traffic (section 3.3). This chapter also describes the steps required to classify the network traffic using ML classifiers (section 3.4). Section 3.5 presents an ML classifier taxonomy based on different properties that define how the classification algorithm works, followed by a description of which ML classifier goes with which kind of data. Section 3.6 begins by justifying the selection of the chosen ML classifiers. It then proceeds to describe the ML classifiers employed in this research, followed by presenting a comparison of ML classifiers. Section 3.7 provides an overview of other DL methods and illustrates how they differ from the ML ones. While section 3.8 reviews the proposed ML methods in the scientific literature for network traffic classification.

3.2 Importance of Traffic Classification

Traffic analysis is the process of analysing the data in the traffic for finding patterns, misconfigurations, relationships, and anomalies. Network traffic classification is a technique used to analyse and classify the traffic into categories such as type of applications, normal or subnormal traffic based on features observed in the traffic according to specific goals of the application

[57, 58]. The phenomenon of linking the network traffic with its applications is known as traffic classification [59].

Network traffic classification plays a vital role in traffic management in computer networks which involves satisfying the QoS requirements of the end-users. Each network has its own QoS requirements and identifying the applications from the traffic is very important to satisfy the Service Level Agreements (SLA) and managing the network resources efficiently. Traffic classification has also importance in the area of troubleshooting where the main functionality is to locate faulty sensors, devices, misconfigurations and locate the point of network errors.

Furthermore, the traffic classification is very useful in the area of security for intrusion detection and avoiding malware from heterogeneous networks [60]. With the emergence of other types of communication architectures such as Internet of Things (IoT) and 5G, applications are generating a large volume of traffic with more stringent QoS requirements, thus a more accurate network traffic classification technique is required compared to traditional classification techniques [61, 62].

3.3 Traffic Classification Techniques

3.3.1 Port-Based Method

The traditional traffic classification techniques involve port-based and payload-based methods. The port-based traffic classification technique extracts the value from packet header and identifies the port numbers for many applications. User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) communicates between the end users by using the port numbers of different flow connections.

Most of the applications have a known port identification number used for local host communications for example, Simple Mail Transfer Protocol (SMTP) is used for sending emails at well-known port number 25 [63].

Initially, during the TCP three-way handshaking mechanism, the classifier residing at the middle of the network looks for the TCP SYN packets to know the information of a new client-server TCP connection at the server side. The application is then searched in the TCP SYN packets destination port number that are assigned by Internet Assigned Numbers Authority (IANA) [64]. The UDP utilises a similar approach for identifying the port number without performing the three-way hand shaking mechanism.

This approach performs well with massive network traffic. However, some P2P applications such as Kazaa and Napster have not registered their port numbers with the IANA. Or applications may use other port numbers than its registered port number to avoid the operating system access control restrictions. In some cases, the port numbers are assigned dynamically for e.g., real video streamer allocates dynamic port numbers for the transferring of data [65].

As a result, the port-based traffic classification approach fails to perform in all above scenarios. The experimental results from the literature show that port-based techniques are not efficient. In [66] authors employed the port-based approach for network traffic classification and found that no more than 70 % accuracy is achieved by utilising the port addresses of IANA list. The researchers in [67] proposed a port-based traffic classification technique and concluded that the simulation did not accurately predict 30-70% of traffic flows.

3.3.2 Payload-Based Method

To address the issue of the port-based traffic classification approach, the payload-based technique is introduced and is also known as Deep Packet Inspection (DPI). The DPI compares the features extracted from packets with a set of characteristic signatures and features to identify different application protocols. This approach is specifically designed for P2P applications [65, 68].

The tools for the DPI are PACE, L7-filter, NBAR and nDPI. The DPI tools have faced several challenges with the growing number of protocols and new applications. Specifically, the DPI tools need to be updated with the creation of new protocols and applications. If these tools are not regularly updated, the performance in terms of prediction will result in erroneous or unknown signatures. Thus, the list of signatures needs to be updated regularly.

This technique solves the issue associated with port-based approaches; however, this technique is computationally expensive. Furthermore, this technique also needs to update the signature pattern for new applications. Finally, this technique fails when privacy policies deny inspection of the packet content and is problematic in dealing with encrypted traffic [69, 70].

3.3.3 ML Based Traffic Classification

Recently, ML methods have been successfully used for network traffic classification. ML methods have addressed the limitations of port-based and payload-based classification methods [71]. These methods can classify the encrypted traffic accurately. These methods can learn the patterns from network traffic automatically. They can characterise network traffic to respective flows and applications after training [71, 72].

ML methods have been proved as a powerful tool for classifying the network traffic using prior knowledge or statistical information extracted from the raw

traffic in the form of features. ML methods are suitable for classifying the network traffic into specified categories having similar characteristics. Each traffic instance is labelled with the specific network application as per its features without inspecting the packet header and payload directly. The significant features involve the statistical patterns such as network traffic duration, number of packets in a given time, the time between two constructive packets and order of packet arrivals etc. [73].

The general process of ML consists of two phases: the training phase and the testing phase. During the training phase, the ML algorithm is executed by feeding network traffic data for the purpose of identifying and differentiating patterns in the network traffic, the ML algorithm learns the network traffic patterns. After the training phase of the ML model, is used to predict the category of unseen network traffic during the testing phase [71].

ML based network traffic classification process consists of five major steps followed in sequence. The significant steps are: 1) data collection, 2) data pre-processing, 3) feature extraction, 4) model training and 5) performance analysis [74].

ML methods have been proved as faster and accurate methods for network traffic classification in comparison to the conventional port-based and payload-based classification methods. These methods work equally well for encrypted network traffic. However, the accuracy of the ML method relies upon the quality and quantity of the training data in addition to the selection of right features extracted from network traffic. Extraction and selection of appropriate network traffic features play a critical role in the successful discrimination of network traffic patterns. However, this task has become difficult due to the increased complexity of modern networks and their applications [75].

ML methods can be broadly categorised into three categories for network traffic classification in general. The significant categories involved supervised ML methods, unsupervised ML methods, and semi-supervised ML methods [71].

Supervised ML methods require labelling the instances of network traffic. During the training phase the ML classifier infers the model parameters (SVM) or the rules (decision tree).

The trained ML model predicts the label of unknown traffic instance to already learnt labels automatically during the testing phase. Generally, a feature selection or reduction process is applied before using supervised learning for selecting the most relevant features for classifying network traffic. Several ML methods have been used in supervised learning modes such as decision trees, Naive Bayes, K-Nearest Neighbour (KNN), SVM, and neural networks [76].

Unsupervised ML methods are helpful for network traffic classification when label data is not available. These methods are capable of extracting significant traffic patterns from unlabelled training data. Unsupervised ML methods are capable of detecting unknown traffic classes [77]. These methods have been widely used in network traffic classification due to the availability of unlabelled traffic [78].

Several methods have been developed in unsupervised learning, such as clustering. This method classifies the network traffic into similar groups based upon similar characteristics of network traffic. The most commonly used methods involve a K-means clustering method and the DBSCAN clustering method [79]. Clustering methods have been successfully employed for differentiating web and P2P network traffic in [80].

Semi-supervised ML methods offer characteristics of both supervised and unsupervised ML methods. Semi-supervised methods enable the labelling of unlabelled network traffic based upon a limited amount of labelled network traffic. This process is generally known as label propagation [81]. The emergence of new applications and new types of network traffic require the application of semi supervised ML methods for detecting zero-day network traffic [82, 83]. For example, semi-supervised ML methods have been used for network traffic classification in [84]. Glennan et al. [84] used K-means clustering method for clustering the network traffic and passed the detected clusters to the supervised decision tree ML method to label network traffic based upon the available label in the labelled training data set.

Similarly, Ran et al. [85] have also used K-means clustering method followed by KNN as supervised ML method for network traffic classification and labelling the unlabelled network traffic.

3.4 Steps of ML Traffic Classification

ML based network traffic classification process consists of five major steps following in sequence. The significant steps include data collection, data pre-processing, feature extraction, model training and performance analysis. The detail is provided as below.

1) **Data collection:**

Data collection is the important and most significant step in applying ML methods to any problem. The goal of this step is to gather sufficient information regarding the problem at hand. It involves applying different procedure of measuring the data using digital or physical sensing devices. The collected data describes the current status, and it is used to define the benchmark data set. Different types of data samples are gathered under multiple experimental scenarios to define the benchmark dataset [60].

In the case of traffic classification, the data collection step involves the collection of a large representative network data without any bias [86]. Network data may be collected from network sessions, and traffic traces at different network layers depend upon the application requirement. For example, network traffic classification requires collecting packet level data labelled with their respective applications [87]. Data can be collected in two phases, offline and online. The offline phase involves the collection of historical data for the training of the ML model. In contrast, the online phase consists of real time network traffic used as inputs to train the ML model.

2) Feature extraction:

Feature extraction involves driving or computing the significant characteristics of the traffic, representing the current state of the process [60]. This step involves the computation of various metrics that reflect specific characteristics of the collected data. The primary purpose of feature extraction is to compute descriptors for characterising the problem of traffic classification [60, 88]. This phase produces data in form of rows and columns to represent samples along with their labels as the target class if available.

The process of identification of relevant and non-redundant features from the raw training data is the most significant step that unleash the potential of data. It is also known as feature engineering in ML research community. Extracting meaningful features from training data set generally require domain expertise [89]. Therefore, it is a challenging and manual task. The use of Deep Learning (DL) methods can automate the feature extraction process [86, 90].

Several feature types have been defined in literature for network traffic classification. The essential types of features include statistical features, graph-based features, and time series-based features [60].

Statistical features are extracted from network packet flows, assuming the traffic flowing at the network layer. The most common statistical features include flow duration, idle time, length of the network packets, the time between conjunctive packets. Statistical features have unique values for different types of applications.

Graph based features represent the internal composition of networks that enables representation into interconnected graphs [91]. In this scenario, a network is generally assumed as a collection of interconnected nodes representing hosts, and the edges between the nodes representing interaction among hosts. These interactions can be assumed as a communication session for exchanging the network packets between different nodes.

Time series based features represent the sequence of events with respect to time. Interaction among network nodes lies in the order of events such as opening and closing communication sessions, starting or finishing transmission of data etc [92]. Time series features generally represent the relationship between inter arrival time and network packet size of the given floor and enable the characterisation of the applications using time series representation [60].

3) Data pre-processing:

The features extracted from raw training data in the feature extraction step of the ML process may contain some missing values and unknown values [86]. In order to make effective use of extracted features to train the ML model, data pre-processing strategies are applied to clean the data from missing values, unknown values and detecting outliers. These issues of missing value, unknown values, and outliers affect the ML model's performance.

Data pre-processing strategies also transform the data to a standard form by using different normalisation, and aggregation operation on feature values. In

the aggregation process, some features are aggregated into a single feature that becomes more valuable to characterise the network data. The normalisation process transforms the values of a feature to a given range such that 0 to 1 [60, 86].

Another essential step during the pre-processing stage in case of supervised ML is to check the class imbalance. The class imbalance presents a scenario containing instances belonging to one or more classes much higher than the other classes. Imbalanced data set may lead to biased training of ML models. In [93] some strategies have been proposed for dealing with class imbalance problem, over sampling data and under sampling data.

Another significant and optional step involves the selection of relevant features used for the training of ML models [60]. The feature selection process discards the irrelevant and redundant features for reducing the number of features using some techniques like principal component analysis. Feature selection helps to reduce the amount of training data that causes faster training of ML models and avoids the curse of dimensionality problem.

Several methods have been proposed for the feature selection process. These methods can be categorised into three categories, filter methods, wrapper methods, and hybrid methods [94].

Filter methods involve scoring each feature based upon some computed metric to signify the relevance of feature to characterise the target label of the sample. Some methods in the filter category contain Gini index [95], Maximum Relevance Minimum Redundancy (MRMR) [96], Information gain [97], Gain ratio, and Correlation based feature selection [98].

In contrast, wrapper methods involve supervised learning for defining an objective to determine the impact of features sets on the classification accuracy of the model. The features providing the best accuracy are selected as the final

selected feature set for training of the ML model. The essential methods in this category are the genetic algorithm and sequential search methods [99]. Hybrid methods involve the combination of filter and wrapper methods for selecting significant features from the training data set.

4) Model training:

The pre-processing step of ML process generates a dataset that is compatible with the processing of any ML model [60]. Output is saved in the form of rows and columns as CSV file format in general. The model training step involves training the ML algorithm to solve a specific problem such as classification problem, regression problem and clustering problem.

Supervised ML classifiers adjust their parameters for minimising the error between actual and output of the model corresponding to a given input during the training process. Common supervised ML classifiers include decision trees, Naive Bayes, neural networks, KNN, SVM, and Random forest.

In contrast, unsupervised algorithms determine the association between input without prior knowledge of output. Associations are computed in terms of similarity or distance. Supervised models are generally used for classification purposes, whereas unsupervised models are used for clustering purposes. Examples of unsupervised ML models consist of K-means clustering [100], EM clustering [101], and DBSCAN clustering method [102].

Many algorithms take advantage of supervised and unsupervised learning. For example, the semi-supervised method exploits the unlabelled data to train classifiers using clustering methods and labelled data to label the clusters. Several methods have been proposed that combine the output of different classifiers called hybrid method or ensemble methods. Examples of methods in this category are the bagging method, stacking method and boosting method [103].

Generating a well-trained ML model involves two steps, training and tuning the ML model. We need to select an appropriate ML model as per training data size and features of network scenario and problem category. The ML model is trained based upon the training data set with the tuning of the hyperparameters. There exist no well-defined theoretical guidelines for tuning hyperparameters. Generally, it involves searching an ample space to determine acceptable hyperparameters or to apply domain expertise to achieve an optimised set of hyperparameters. After the training process, ML models are validated based upon cross validation strategy for evaluating the accuracy of the ML model [86].

Validation results signify the level of overfitting or underfitting of the ML model. Validation results provide guidelines for optimising a ML model, such that increasing the size of the training data set and reducing the complexity of the model for avoiding overfitting problem [86].

5) Performance analysis:

In order to perform a comprehensive evaluation of ML models, several objective metrics have been defined. Performance analysis step in ML process computes performance metrics quantitatively for evaluating ML models.

Supervised learning models are generally evaluated in terms of the classification performance of the model. Many associations have been identified between ground truth and the prediction of the model.

The most important representation is the confusion matrix that represents the true positives, true negatives, false positives, and false negatives in the form of a matrix. However, the confusion matrix is not directly compared to different ML models. Many metrics can be derived using the information present in the confusion matrix such as classification accuracy, precision, recall or True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate

(FNR), True Negative Rate (TNR), F-measure, area under Receiver Operator Characteristics (ROC) curve, area under Precision Recall (PR) curve and kappa statistics [60, 104].

In the case of a multi-class classification problem, the micro or macro average of the above-mentioned performance metrics can be used to evaluate and compare the performance of ML models. These metrics provide different aspects of the performance of ML models [105, 106].

3.5 Classifier Taxonomy

A number of properties exist that describe each ML classifier. The following four main properties define how the classification algorithm works, what kind of data it requires and how good the classifier is [170, 171]:

1) Generative classifier or discriminative classifier:

Generative classifiers e.g., Naive Bayes, learn models for each class, and to classify a feature vector, generative classifiers compute the likelihood of each class and choose the most likely. In contrast discriminative classifiers e.g., SVM, only learn the way of discriminating the classes in order to classify a feature vector directly [172].

2) Dynamic or Static

Dynamic classifiers have the capability of considering the temporal information during the classification as a sequence of feature vectors can be classified. This involves extracting features from different time segments in order to build a temporal sequence of feature vectors and feed into the dynamic classifier. so dynamic classifiers e.g., hidden Markov model, possess the ability to detect or catch the relevant temporal variations present in the extracted features. whereas static classifiers e.g.,

MLP do not catch the relevant temporal variations during the classification as they classify a single feature vector [172, 179, 183].

3) Stable or Unstable

Stable classifiers e.g., Linear Discriminant Analysis (LDA), SVM, etc., are robust against the small changes or variations in the training set, and their performance is not significantly impacted. In the contrast, unstable classifiers e.g., MLP, decision tree, etc., are susceptible to the small variations in the training set which may lead to considerable changes in the constructed classifier [173, 174].

Training a classifier can often be divided into several stages in an attempt to minimise the bias-variance trade-off problem. This first starts by using a training set to estimate the classifier's decision boundary, followed by a validation set, which is used to test and further refine the classifier's decision boundary [174]. There is a natural trade-off between bias and variance, and to achieve a low classification error both bias and variance must be low. Stable classifiers tend to have a high bias and a low variance, whereas a low bias and high variance are observed in unstable classifiers [179, 184].

Stable classifiers perform better than unstable classifiers in the presence of non-stationary features that vary rapidly or frequently over time or sessions i.e., training sets coming from different sessions [179, 185].

4) Regularised classifier

Regularisation is the process of controlling or regularising the complexity of a classifier in order to prevent overfitting. Thus, the regularised classifier is more robust against outliers [173].

3.5.1 Which Classifier Goes with which Kind of data

- **Noise and outliers**

An outlier is defined as a data point that is different from the remaining data. Whereas noise can be defined as errors in the values of attributes or mislabelled class values [186]. So regularised classifiers e.g., the regularised version of LDA (RLDA), Linear SVM, deal better than an unregularised version of LDA in the presence of outliers and errors in the training set. Similarly, a nonlinear SVM is more appropriate for dealing with outliers and errors than an unregularised MLP. As they use a regularisation parameter that enables accommodation to outliers and allows errors on the training set. This results in increasing the generalisation capabilities of the classifier [179, 185].

- **High dimensionality**

SVM is one of the most appropriate classifiers to handle the feature vectors of high dimensionality, for example, features extracted from a number of channels and from a number of time segments before combining them into a single feature vector. Moreover, dynamic classifiers also have the capacity to cope well if high dimensionality is due to a large number of time segments, as they have the ability to deal with sequences of feature vectors at a time instead of dealing with a single feature vector of high dimensionality. KNN should not be used with high dimensional feature vectors due to the sensitivity to the curse of dimensionality, but it can be efficient with a small number of features [179, 185].

- **Time information**

Dynamic classifiers possess the ability to efficiently exploit temporal information contained in features. Moreover, combining classifiers over time can also be efficient in utilising the time information [179, 185].

- **Non-stationarity**

In the presence of non-stationary features, one of the efficient ways of dealing with this issue is a combination of classifiers as it reduces the variance. Stable classifiers such as SVM or LDA can also be applied in this context, but this would be outperformed by combining classifiers [179, 185].

- **Small training sets**

In the presence of small training sets, simple classifiers with few parameters such as LDA should be considered [179, 185].

3.6 ML Classifiers

3.6.1 Overview

This section describes the ML classifiers employed in this research for classifying smartphone applications' network traffic. There are a variety of ML classifiers that exist; however, they can still be categorised into five main tribes: Bayesians, Evolutionists, Connectionists, Analogists and Symbolists [109, 125].

Bayesians are influenced by statistics and focused on using probabilistic inference to evaluate a hypothesis where certain outcomes are more likely than others. Naive Bayes is an example of an ML classifier that belongs to the Bayesians tribe. While evolutionists are influenced by biology and natural

selection and they focused on using the principles of natural selection to produce dependents that are most successful within their environment. genetic algorithms are an example that belongs to this type of tribe [109, 125].

Moreover, from the following three tribes; connectionists, analogists and symbolists, five commonly used ML classifiers with distinct learning approaches were considered in this research. These ML classifiers are MLP, KNN, decision tree, Random forests and SVM.

For example, MLP is one of popular Artificial Neural Networks (ANNs) that is inspired by biological neural networks. MLP belongs to the connectionists tribe that is formed of linked nodes and directed linkages. wherein each node performs a biased weighted sum of its inputs and applies an activation function to transfer its output to the next layer.

While both KNN and SVM belong to the analogists ML tribe, as they focus on identifying similarities between situations and thereby inferring other similarities. Thus, the learning rests upon building analogies between available data. However, they differ in their learning approaches. SVM is considered an eager learner, as the model is constructed from the training data before classifying the unseen testing data. Whereas KNN is known as a lazy learner. Unlike SVM, KNN has no explicit training phase. Instead, the learning phase is deferred till the test cases are executed against the model.

Finally, decision trees and random forests belong to the symbolists ML tribe, where knowledge is built by constructing symbolic representations of a concept. However, the main distinction between decision tree and random forests is that the random forest is an ensemble-based learning method that comprises multiple decision trees and is formed by using a bagging technique along with a randomised selection of features.

3.6.2 Artificial Neural Networks: Multilayer Perceptron (MLP)

MLP is a feed forward neural network consisting of different layers of interconnected neurons. It contains three layers: the input layer, hidden layer, and output layer containing different neurons in each layer. Each neuron performs a biased weighted sum of its inputs and applies an activation function to transfer its output to the next layer. MLP can model any arbitrary complexity with a number of layers and the number of units in each layer. During the training process, weights are optimised to obtain minimum error at the output layer [107].

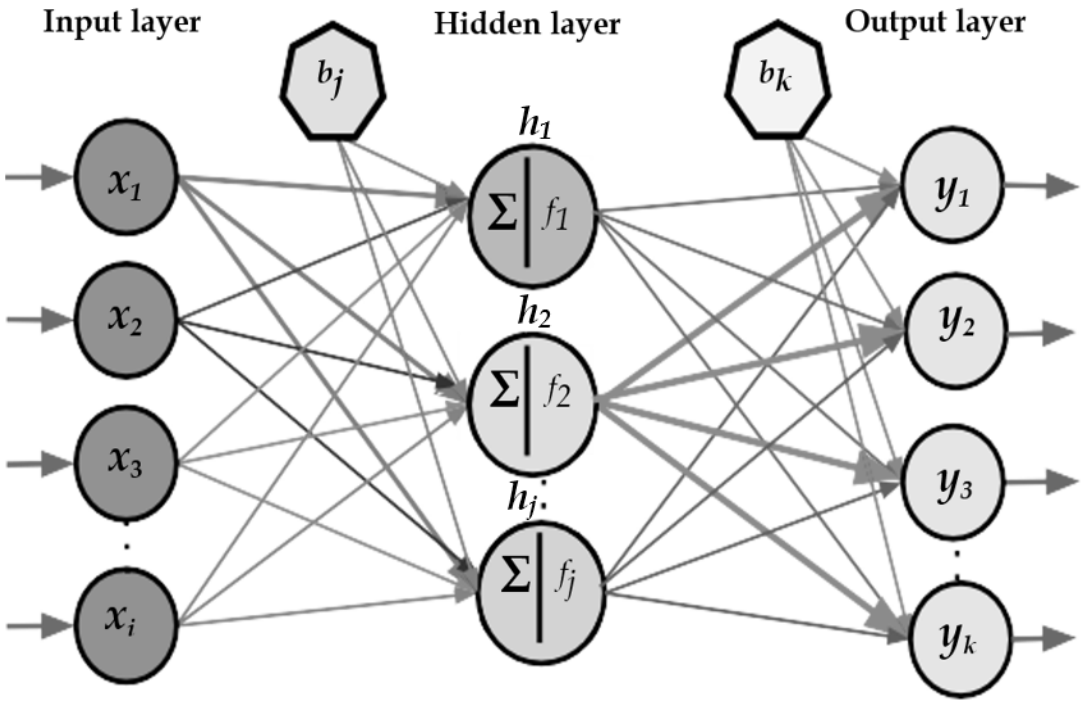


Figure 3. 1: Multilayer Perceptron

Figure 3. 1 depicts an MLP network with a single hidden layer. The first layer is the input layer which takes each training instance and passes through the input neurons unchanged. Each neuron or node in each layer is connected to all nodes of the following layer, and the connection between the nodes is associated with weighting values ranging from [-1.0 to 1.0], or [-0.5 to 0.5]

[141]. Nodes of each layer except of the input layer, also have a constant input called a bias that is added to the associated weights. Each neuron of an MLP has two functions, summation, and activation functions. The summation function is used to obtain the product of the input values, values of associated weights and bias [108].

$$s_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \qquad \textbf{(3.1)}$$

Equation 3.1 represents the summation function s_j of the hidden neuron h_j, where n denotes the total number of the inputs, x_i is the input variable, b_j is the bias value of the hidden nodes, and w_{ij} refers to the connection weight between the node i of input layer to node j of the hidden layer.

The next step is the application of the activation function, so the result of the computation in equation 3.1 of the summation function is passed onto an activation function to produce the output of the neuron [169]. An activation function is also known as a squashing function because it restricts the output values of neurons in hidden and output layers onto two small values. A number of activation functions exist and can be utilised in MLP, these include linear function, heaviside step function, gaussian function, etc [107, 141]. The most common is the S-shaped sigmoid function, which can be stated as follows:

$$f_j(s_j) = \frac{1}{1 + e^{-s_j}} \qquad \textbf{(3.2)}$$

Therefore, the final output of neuron h_j in hidden layer is:

$$h_j = f_j \left(\sum_{i=1}^{n} w_{ij} x_i + b_j \right) \quad \textbf{(3.3)}$$

And the predicted output of node y_k in the output layer is calculated as follows:

$$y_k = f_k \left(\sum_{j=1}^{m} w_{jk} h_j + b_k \right) \quad \textbf{(3.4)}$$

where m is the total number of hidden nodes, f_k is the activation function of node y_k of output layer, b_k is another bias value for nodes in output layer, and w_{jk} refers to the connection weight between the node j of hidden layer to node k of output layer.

During the supervised training of an MLP, weights and biases are learned usually using the backpropagation algorithm to approximate an unknown input-output relation. Therefore, the objective is to minimize the difference between the network's prediction and the actual output [107].

An MLP without a hidden layer is called a perceptron and is applied to classify the linearly separable data, but it is not well suited for nonlinear cases. whereas an MLP solves this issue and is applied to classify the data that are not linearly separable. This is because each neuron in the hidden and output layers has a nonlinear activation function allowing an MLP to distinguish data that is not linearly separable [187, 188].

MLP is the most popular and widely used neural network architecture which has been successfully applied in solving a wide variety range of classification and regression problems [175, 176].

An MLP is known as a universal approximator the strength of an MLP lies in the fact that is capable to approximate any smooth function to any desired degree of accuracy as long as the number of hidden layer neuron increases [177]. However, this makes a classifier sensitive to overtraining which causes overfitting, meaning that the classifier tends to memorise the training data causing it to generalize poorly on unseen data [178, 179].

One of the most common limitations of an MLP that restricts and complicates its application lies in its knowledge representation. This is known as black box limitation; the weights of an MLP provide no explicit information that users could be able to interpret, thus, it is difficult to acquire explicit information about the underlying function implemented by an MLP [180].

Another common limitation of an MLP is associated with the optimisation algorithm. The backpropagation algorithm during the weights and biases adjustment to minimise the prediction error using the gradient descent method does not guarantee to find the globally optimal set of weights and biases during the training. As a result, becomes trapped in local minima instead of finding the global minimum [181, 182].

3.6.3 Lazy Learner: K-Nearest Neighbour (KNN)

KNN is a supervised machine learning method for solving classification and regression problems. It works by analysing the distance between input data samples. KNN is also known as a non-parametric lazy learning algorithm. Non-parametric means the learning algorithm makes no assumptions about the structure or distribution of the underlying data. Being a lazy learner means there is no explicit learning phase, instead the learning phase deferrers till the

test cases are being executed against the model [117]. KNN is also called an instance-based learning algorithm, as it stores the training set and for classifying a new unclassified instance or test record, it firstly searches through the training set for the most similar instances (k nearest neighbours) using a distance measure function, and then classifies the test record according to the majority class among k nearest neighbours [118].

To define which of k nearest neighbours in the training set are the most similar to the new test sample, a distance measure function is used. various distance functions have been used in the literature such as Euclidean Distance, Hamming Distance, Manhattan Distance, etc., among these distance measure functions; Euclidean is the most popular and widely used one [189, 190]. Euclidean distance is calculated as the square root of the sum of the squared differences between the elements of two factors [191].

The following paragraphs will illustrate how KNN classifies a new test sample using the Euclidean distance. Table 3.1 shows a small dataset of 5 records of student results labelled as pass or fail (binary classification), with two input variables: 1- mathematics 2- computer science [118, 191].

Table 3. 1: Small training set of student results

Mathematics	Computer science	Result (Pass/Fail)
4	3	Fail
6	7	Pass
7	8	Pass
5	5	Fail
8	8	Pass

The objective here is to classify a new unlabelled test sample given in table 3.2 into class pass or fail.

Table 3. 2: An unlabelled testing set

Mathematics	Computer science	Result (Pass/Fail)
6	8	??

The training set in table 3.1 contains only two features, so the KNN algorithm treats the features as coordinates in a two-dimensional feature space.

The first step is to calculate the Euclidean distance between the new test sample or instance in table 3.2 and all instances in the training set in table 3.1 based on the following Euclidean distance formula:

$$\text{Euclidean distance} = \sqrt{(x_{o1} - x_{a1})^2 + (x_{o2} - x_{a2})^2} \quad \textbf{(3.5)}$$

where o represents the observed value that given in test sample, and a represents the actual value in a training set.

Table 3.3 lists the Euclidean distances between each training instance and the new unlabelled test sample.

Table 3. 3: Euclidean distances for training data to the new unlabelled instance

No	Euclidean measure	Squared difference	Sum of squared differences	Square root of the sum	Distance
1	$d = \sqrt{(6-4)^2+(8-3)^2}$	$(6 - 4)^2 = 2$ $(8 - 3)^2 = 5$	$(2)^2 + (5)^2$ $= 4 + 25 = 29$	$\sqrt{29}$	5.38
2	$d = \sqrt{(6-6)^2+(8-7)^2}$	$(6 - 6) = 0$ $(8 - 7) = 1$	$(0)^2 + (1)^2$ $= 0 + 1 = 1$	$\sqrt{1}$	1
3	$d = \sqrt{(6-7)^2+(8-8)^2}$	$(6 - 7) = -1$ $(8 - 8) = 0$	$(-1)^2 + (0)^2$ $= 1 + 0 = 1$	$\sqrt{1}$	1
4	$d = \sqrt{(6-5)^2+(8-5)^2}$	$(6 - 5) = 1$ $(8 - 5) = 3$	$(1)^2 + (3)^2$ $= 1 + 9 = 10$	$\sqrt{10}$	3.16
5	$d = \sqrt{(6-8)^2+(8-8)^2}$	$(6 - 8) = -2$ $(8 - 8) = 0$	$(-2)^2 + (0)^2$ $= 4 + 0 = 4$	$\sqrt{4}$	2

For this case, the value of k is set to 3 for the KNN algorithm, so that the new unlabelled test sample would be classified according to the smallest distance or closest 3 data points or neighbours to it.

Thus, The K = 3 most similar neighbours with minimum distances to the new unlabelled test instance are: no 2, no 3, and no 5. Now, we apply the majority of voting technique and select the majority class in the neighbours. In this case, all the three neighbours have the same class label of pass. Therefore, the new unlabelled test instance will be classified as pass.

The value of k (number of neighbours) is considered an important hyperparameter that plays a crucial role in the KNN algorithm, when the value of k decreases or increases, a major change in the outcomes of the KNN classifier can be noted. For the classification, it is recommended to select a k with odd values to avoid a tie in the voting phase. Decreasing the k value e.g., k = 1, might lead to misclassification, particularly in the presence of noisy samples. It could be possible that the nearest neighbour of this particular sample is one of the noisy samples, resulting in a wrong prediction. Moreover, a smaller k value = 1, could sharpen the boundaries and might lead the classifier towards overfitting, tending to memorise the training set at the cost of generalisability. In contrast, k with large values is more robust to noise due to the contribution of more neighbours during majority voting. So, it is more likely a classifier keeps making more accurate predictions when the value of k increases till a certain k value, in which after that certain point, classifier's accuracy starts to decrease, thus this would be the point of an optimal k value [117, 192].

In some cases, it is common that some features are more relevant than others, when the number of irrelevant features increases, the distances computed in the KNN classifier will be dominated by these features. This is known as the curse of dimensionality. Generally, most ML classifiers suffer from irrelevant

features, as a result, they perform poorly. But KNN is more sensitive to the curse of dimensionality than other classifiers in the presence of irrelevant features and its performance is heavily degraded with a large number of features in comparison to other classifiers [117, 193].

KNN is fast in training as it does not require building a model, but for classifying a new instance, the distance between the new instance and all instances in the training set must be calculated, which makes the KNN slow in the testing phase and this becomes significantly slower as the number of examples increases [117].

3.6.4 Decision Tree

Decision tree is a supervised ML method that involves building tree shaped graph to predict possible output corresponding to input values. The built tree contains one root element and some internal elements called decision nodes, used to test the input against a learnt expression. The leaf nodes of the tree correspond to the final prediction of the classifier. The decision tree is used to drive decision rules for solving the decision problem by starting at the root node and moving downward to word leaf nodes to predict the target class. Many variants have been proposed in decision trees classifiers such as Iterative Dichotomiser 3 (ID3), and C4.5 [110].

3.6.5 Ensemble-Based Learner: Random Forest

Random forest is a supervised learning algorithm that is used for solving both classification and regression problems. Random forest is a popular ensemble-based learning method that is formed by using a bagging technique along with a randomised selection of features.

Bagging, which stands for bootstrap aggregating is an ensemble learning technique in which a homogeneous group of individual learners of the same

type also known as base learners (e.g., decision trees) are trained independently and in parallel then their predictions are combined. For the classification, a voting method is adopted whereas averaging method is applied for regression [194].

The bagging process is described as follows: given the original data set D consisting of m samples and since bagging employs bootstrap sampling method, thus one sample is randomly picked from the original data set D and copied into a sampling set D' and then placed back into the original data set D so it has the opportunity to be picked once again and copied into D'.

So, by repeating the process of sampling with replacement m times, a bootstrap sampling data set D' is constructed consisting of m samples. Due to sampling with replacement, a number of samples may be repeated in D' while other samples from the original data set D may never appear in D' even if the size of D' data set is equivalent to the original data set D. So, the probability of a particular sample not being selected from D in any of m rounds is $(1 - 1/m)^m$.

$$\lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.368, \tag{3.6}$$

This means that around 36.8 % of the original samples are not included in D'.

Furthermore, applying the process of constructing a bootstrap sampling data set D' for T times results in T data sets where each consists of m bootstrap samples. Next, the base learners are trained on these data sets, then in the test stage the results from the base learners are combined and the final prediction for the classification is made by conducting the majority voting among the base learners. Moreover, in case of multiple classes have the same number of votes, one can be chosen at random, or confidence of votes can be further investigated [194, 199].

Since the effectiveness of ensemble learning depends on whether or not all individual learners are diverse enough, therefore the main idea is to enhance the diversity between the individual learners which is achievable by introducing some randomness into the learning process. Thus, bagging introduces diversity through data sample manipulation which is particularly useful with unstable learners that are sensitive to training data manipulation, thus when bagging is applied to unstable learners such as decision trees it helps in reducing the variance which alleviates overfitting [194, 111].

Random forest is an extension of bagging and is constructed by using the bagging technique along with the randomised selection of features.

Random forest further enlarges the diversity among the base learners by employing both data sample manipulation and input feature manipulation. Thus, each tree in the Random forest is built from a different random subset of the features which is also known as subspace sampling. The random sampling of the features avoids the domination of some strong features that have more predictive power for the output class, as no matter how many bootstrap samples are used, stronger features will be selected alone in many trees leading to high tree correlation. Therefore, Random forest helps in reducing the correlation between the trees in the ensemble by adding extra randomness into the tree-growing process. More specifically, instead of searching for the best feature when splitting a node, the algorithm searches the best feature only across randomly selected features from the feature set [115, 217].

Hence, adding the random selection of features on top of bagging encourages diversity and leads to a better generalisation ability. Additionally, the training time of each tree is reduced [194, 199].

Each tree in the Random forest is constructed based on the following steps [115, 194]:

Step 1: Given the original data set D, follow the bootstrap sampling method with replacement by randomly picking samples from the original data set D copying them into sampling set D' and placing them back into D. The size of the sampling set D' is adjustable parameter, the most common choice is $|D'| = |D|$.

Step 2: Train the decision tree on D' with one major modification, instead of evaluating all the features to find the best split point to split the node, consider only the randomly selected features m from the feature set p.

The size of the random subset of features m that is considered at any given split is also a free parameter. Typically for the classification problem, the most common choice is using the square root of the total number of features $m=\sqrt{p}$.

For the classification, based on the majority voting method where an ensemble consists of T decision trees $\{h_1, h_2, \ldots, h_T\}$, where h_i predicts the class label $\{c_1, c_2, \ldots, c_N\}$, given a test sample x, $h_i^j(x)$ is the output of h_i on test sample x for the class label c_j.

$$H(X) = C \underset{j}{\operatorname{argmax}} \sum_{i=1}^{T} h_i^j(x). \quad \textbf{(3.7)}$$

No particular value type assumed in the above equation, however, in case of assuming the class label $h_i^j(x) \in \{0, 1\}$: then the output would be 1 in case of h_i predicted the class label as c_j and otherwise 0. Figure 3.2 illustrates the graphical formulation of Random forest.

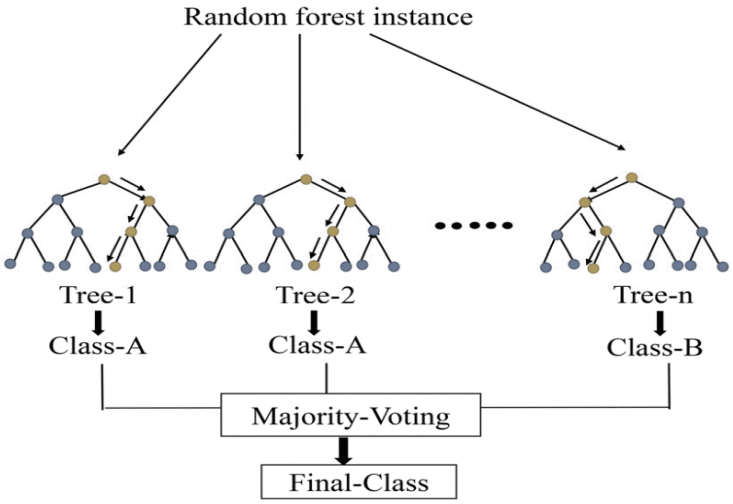


Figure 3. 2: Random forest [116]

In addition to the advantage of diversity generation mechanisms employed by the Random forest to enhance the diversity among the base learners which consequently leads to a better generalisation ability. Another advantage of using Random forest is its ability to measure the importance of each feature across all trees in the forest in terms of their contribution to the classification and then sorting the features in descending order according to their predictive power for the output class [208, 115].

However, in terms of disadvantages, Random forest is less interpretable by users and requires more time to build compared to a single tree classifier. This is due to a large number of trees being used as base learners in which each tree is heavily influenced by random selections of samples and features [207, 218].

3.6.6 Support Vector Machine (SVM)

SVM is used for both regression and classification tasks, but it is more commonly applied in solving classification problems [117]. Despite the ability of SVM to perform linear classification, moreover, in cases when the original data set is linearly non-separable in the original input space, SVM uses the kernel trick to transform data from the original input space into higher

dimensional feature space. Once this transformation is achieved in this feature space, a linear hyperplane is obtained to separate the different classes involved in the classification task [111].

The goal of the SVM is to construct a hyperplane, i.e., a decision boundary that has a maximum margin (distance) between samples of different classes. Support vectors are the nearest data points to the hyperplane that affect the position and the orientation of the hyperplane and based on these support vectors, classifier's margin is maximised [117].

3.6.6.1 Linear SVM

Assuming the training data is linearly separable, the basic concept of the classification is to find a separating hyperplane that separate the samples of different classes, given a training set D defined by the following:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}, y_i \in \{-1,+1\}. \quad \textbf{(3.8)}$$

Where y_i is either 1 or -1 for positive and negative classes. However, a number of separating hyperplanes possibly could be constructed as shown in figure 3.3. In comparison to other hyperplanes, the hyperplane in red is precisely right in the middle of the two classes and it has the maximum separation distance from all training samples, and that is the one that should be chosen. Moreover, the red hyperplane is more robust against noises in data, as is less likely to be affected in the presence of small perturbations in the data, thus it provides better generalization capability [194].

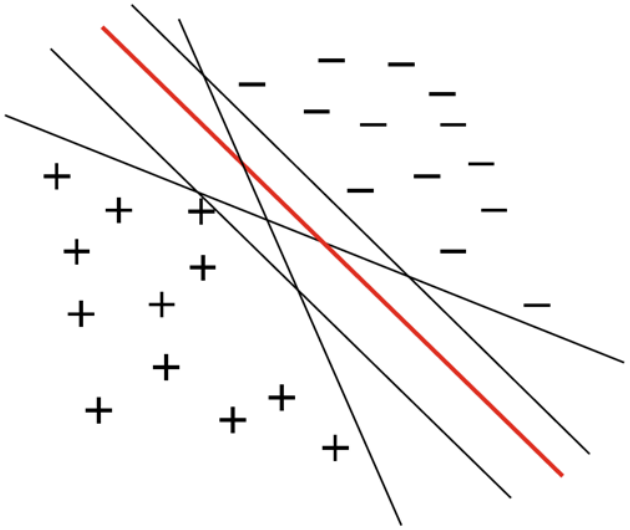


Figure 3. 3: Multiple hyperplanes separating samples of two classes [194]

The hyperplane can be expressed as the following linear function:

$$w_1x_1 + w_2x_2 + \cdots + w_dx_d + b, \quad \textbf{(3.9)}$$

where w_d is a coefficient or weight, x_d is the input variable, and b refers to the bias [194, 195].

Or commonly written in the vector form

$$w^T x + b = 0, \quad \textbf{(3.10)}$$

where $w = (w_1;w_2; \ldots ;w_d)$ is the normal weight vector that controls the direction of the hyperplane, x is input vector and b is the bias offset which controls the distance between the hyperplane and the origin. So, w is slope and b is the intercept which determine the separating hyperplane [194], and the distance from any sample x to a hyperplane can calculated as:

$$\frac{|w^T x + b|}{||w||} \quad \textbf{(3.11)}$$

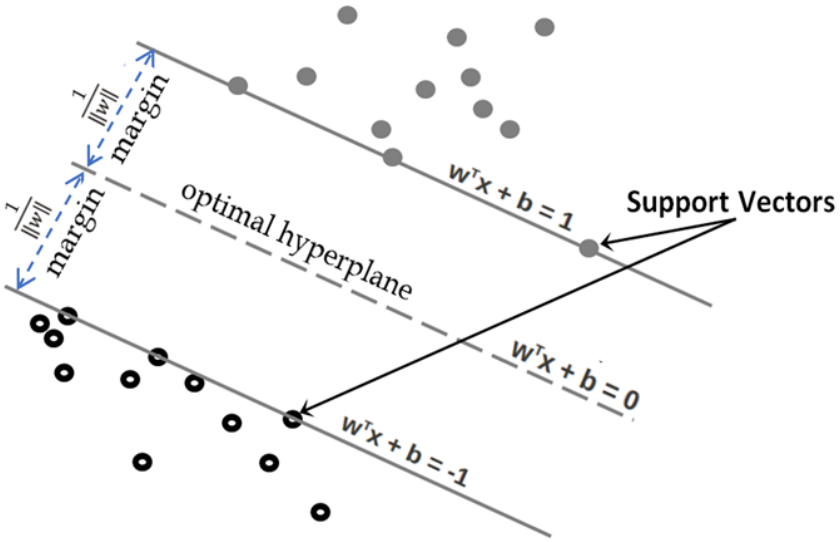


Figure 3. 4: SVM

As depicted in figure 3.4, if the hyperplane $(\boldsymbol{w}, \boldsymbol{b})$ perfectly separates the training data points for $\forall\ (x_i, y_i) \in D$, there is:

$$\begin{cases} \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} + b \geq +1, & y_i = +1, \\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} + b \leq -1, & y_i = -1, \end{cases} \tag{3.12}$$

The nearest data points to the hyperplane called support vectors in which equality holds for in (3.12), and the total distance from support vectors of two different classes to the hyperplane called margin [111, 194]. Expressed as

$$\gamma = \frac{2}{\|w\|} \tag{3.13}$$

The next step is to find a maximum-margin hyperplane it also means finding the parameters w and b that maximise γ subject to the constraints in (3.12), that is:

$$\underset{w,\,b}{max}\ \frac{2}{\|\boldsymbol{w}\|} \tag{3.14}$$

s.t $y_i(w^T x_i + b) \geq 1, \quad i = 1, 2, \ldots, m.$

To maximize the margin is the same as minimising $||w||^2$ thus (3.14) can be rewritten as:

$$\min_{w,\, b} \ \frac{1}{2} ||w||^2 \qquad \textbf{(3.15)}$$

s.t $y_i(w^T x_i + b) \geq 1, \quad i = 1, 2, \ldots, m.$

This is known as the primal form of SVM. (3.15) is a convex quadratic programming optimisation problem which can be approached by introducing Lagrange multipliers and based on Karush–Kuhn–Tucker (KKT) conditions [194, 196]. Adding the constraints with Lagrange multiplier $a_i \geq 0$ for each training example in (3.15) gives the Lagrange function:

$$L(w, b, \alpha) = \frac{1}{2} ||w||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i(w^T x_i + b)) \qquad \textbf{(3.16)}$$

Where $\alpha = (\alpha_1; \alpha_2; \ldots; \alpha_m)$. Taking the partial derivatives of L(w, b, α) with respect to w and b and setting those to zero, we obtain

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \qquad \textbf{(3.17)}$$

$$0 = \sum_{i=1}^{m} \alpha_i y_i \qquad \textbf{(3.18)}$$

Furthermore, by Substituting (3.17) into (3.16) w is eliminated from $L(w, b, \alpha)$. Then with the constraint (3.18), the problem can be reformulated to the dual form problem expressed as follows:

$$\max_{\alpha} \ \sum_{i=1}^{m} \alpha - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \, x_i^T x_j$$

$$\sum_{i=1}^{m}$$

$$\text{s.t.} \begin{cases} \sum \alpha_i y_i = 0 \\ \alpha_i \geq 0, \quad i = 1,2,\ldots, m. \end{cases} \tag{3.19}$$

Once the above dual optimisation problem is solved, we can obtain α of Lagrange multiplier in (3.16) corresponding to training data points (x_i, y_i), and subsequently w and b are obtained [194, 197]. Then, the desired model is obtained:

$$\begin{aligned} f(x) &= \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + \mathrm{b} \\ &= \sum_{i=1}^{m} \alpha_i y_i x_i^{\mathrm{T}} x + b. \end{aligned} \tag{3.20}$$

Since (3.15) is an optimisation problem with inequality constraints. The KKT must be satisfied.

$$\begin{cases} \alpha_i \geq 0; \\ y_i f(\boldsymbol{x}_i) - 1 \geq 0; \\ \alpha_i (y_i f(\boldsymbol{x}_i) - 1) = 0. \end{cases} \tag{3.21}$$

Thus, for any training point (x_i, y_i) we either have data points whose corresponding Lagrange multiplier $\alpha_i > 0$ or $\alpha_i = 0$. When $\alpha_i > 0$ or $y_i f(x_i) = 1$, these data points are the support vectors, thus the maximum-margin hyperplane depends only on these points which are used in the summation in (3.20). While the rest of data points whose corresponding Lagrange multiplier $\alpha_i = 0$ are not the support vectors and have no impact on maximum-margin hyperplane, so removing them once the training is completed, results in the same maximum-margin hyperplane for the linear SVMs as the final model only depends on the support vectors [111, 194, 195].

As the final solution of SVMs relies on a small number of training data points, SVMs are also called sparse models or machines. Sparse models are usually not Susceptible to limitations such as outliers and overfitting [111].

Finally, to compute the bias b, we can observe that there is $y_s f(\boldsymbol{x}_s) = 1$ for every support vector $(\boldsymbol{x}_s, y_s)$, that is

$$y_s \left(\sum_{i \in S} a_i y_i \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_s + b \right) = 1, \tag{3.22}$$

It is possible to obtain the b by substituting any support vectors into the equation (2.22), however it is more convenient to average over all support vectors and compute b as:

$$b = \frac{1}{|S|} \sum_{s \in S} \left(\frac{1}{y_s} - \sum_{i \in S} a_i y_i \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_s \right) \tag{3.23}$$

Where S denotes for index set of all support vectors $S = \{i \mid a_i > 0, i = 1, 2, \ldots, m\}$ is the index set of all support vectors [194, 198].

3.6.6.2 Soft SVM and Regularisation

The SVM formulation discussed previously is known as the hard margin SVM, which is only suitable for linearly separable data. However, in situations when the training data are not completely separable [196], the maximum-margin hyperplane may not exist [111].

The hard margin requires all training data points to be correctly classified subject to the constraints (3.12), whereas the soft margin allows violation of the constraint

$$y_i(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}_i + b) \geq 1, \tag{3.24}$$

To be specific, the soft margin aims to maximise the margin and simultaneously tries to minimise the number of data points violating the constraint. In this case, nonnegative slack variables $\xi i \geq 0$ are introduced for each training data point to allow some of data points to be inside the margin or even at the wrong side of the hyperplane [199].

Thus, each training data point has a corresponding slack variable that indicates the level to which the constraint (3.24) is violated [194].

The primal form of SVM in (3.15) is extended by adding the slack term to the objective function resulting in the following soft margin optimisation problem [111]:

$$\min_{w,\, b,\, \xi i} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{m} \xi i \qquad \textbf{(3.25)}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x_i} + b) \geq 1 - \xi i$$

$$\xi i \geq 0, \qquad i = 1, 2, \ldots, m.$$

Where the regularisation term C is a hyperparameter that controls the trade-off between the margin maximization (corresponding to minimising $||w||$ 2 /2) against slack variable minimisation (corresponding to minimising the sum of the slack terms) [195. 199]. By increasing the C value, a tighter margin is obtained, and more effort is made on minimising the number of misclassifications. In other words, large values of C forces all the training data points to obey the constraint (3.24) which is equivalent to (3.15) that is the hard margin [194, 196].

On the other hand, decreasing the C value result in relaxed margin, and permits some data points to violate the constraint in order to achieve a large margin[194, 196]. Figure 3.5 shows the effect of large and low values of the regularisation parameter and margin violation, C_2 has a tighter margin and

fewer training data points are within the margin, whereas C_1 has a wider margin.

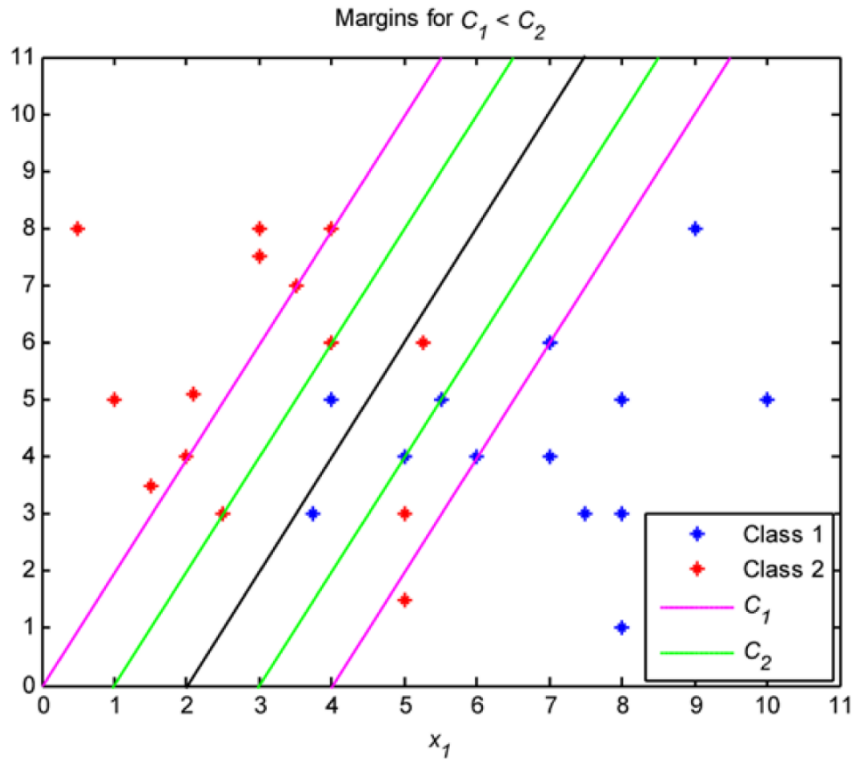


Figure 3. 5: the effect of large and low values of the regularisation parameter C [196]

The problem in (3.25) again is a quadratic programming problem similar to (3.15), hence, to resolve the optimisation problem in (3.25) the same procedure of resolving (3.15) is followed [111].

The soft-margin dual problem is almost identical to the hard-margin dual problem presented in (3.19), the only difference is that each dual variable is currently upper bounded by the regularization parameter C ($0 \leq \alpha_i \leq C$) [111, 196, 197].

3.6.6.3 Nonlinear SVM: the Kernel Trick

In situations when the training data are not linearly separable in the original input space it may become linearly separable in a higher dimensional space.

In such cases training data are mapped from the original input space into a higher dimensional feature space in which the linear classification can be applied [111].

Let φ represent feature vector x that is mapped into the feature space, so the separating hyperplane model can be represented as follows in the feature space:

$$f(x) = w^{T} \varphi(x) + b, \tag{3.26}$$

And its dual formulation is

$$\max_{\alpha} \sum_{i=1}^{m} \alpha - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \varphi(x_i)^{T} \varphi(x_j)$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{m} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C, \quad i = 1,2,\ldots, m. \end{cases} \tag{3.27}$$

To solve (3.27) it is required to calculate the inner product $\varphi(x_i)^{T} \varphi(x_j)$ of the mapped feature vectors of x_i and x_j. However, since the direct calculation of $\varphi(x_i)^{T} \varphi(x_j)$ is difficult as the mapped feature space could have high or even infinite dimensionality [194]. Therefore, we assume there exists a function denoted as:

$$k(x_i, x_j) = (\varphi(x_i), \varphi(x_j)) = \varphi(x_i)^{T} \varphi(x_j), \tag{3.28}$$

This indicates that by the application of the function $\kappa(\cdot, \cdot)$ the inner product of vectors of x_i and x_j in the feature space can be calculated in the input space using the kernel function, so there is no need to perform the calculation of the inner product in the feature space (this process is called kernel trick) [194]. Thus (3.27) can be rewritten as:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j k(x_i , x_j)$$

$$\text{s.t.} \quad \begin{cases} \sum_{i=1}^{m} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C, \quad i = 1,2,\ldots, m. \end{cases} \qquad \textbf{(3.29)}$$

Solving it gives

$$\begin{aligned} f(x) &= w^{\mathrm{T}} \varphi(x) + b \\ &= \sum_{i=1}^{m} a_i y_i \varphi(x_i)^{\mathrm{T}} \varphi(x) + b \\ &= \sum_{i=1}^{m} a_i y_i k(x , x_i) + b, \end{aligned} \qquad \textbf{(3.30)}$$

where $k(\cdot , \cdot)$ represents the application of a Kernel function to the SVM, from (3.30) we notice that by applying the kernel functions the optimal solution could be expanded by training data points, and this expansion is so-called support vector expansion [194].

A function could be specified as kernel function $k(\cdot , \cdot)$ if the details of mapping $\varphi(.)$ are explicit. However, $\varphi(.)$ usually are unknown, thus for a function to be a valid kernel function it must satisfy a certain condition known as Mercer's theorem condition. Thus, according to the Mercer's theorem a valid kernel function must be symmetric with a positive semi-definite kernel matrix [111, 194].

Let X denote the input space, and $k(\cdot , \cdot)$ is a symmetric function on $X \times X$. Then k is a valid kernel function if and only if its corresponding kernel matrix is positive semi-definite for any training set [111, 194].

Therefore, for every positive semi-definite kernel matrix there always would be a corresponding mapping φ meaning that every kernel function implicitly

specifies a feature space, but we may not know explicitly the details of the feature mapping. For this reason, the selection of kernel is the biggest uncertainty for SVMs. Because selecting a poor kernel would map the training data into a poor feature space resulting in poor performance [194].

Table 3.4 shows different kernel functions commonly used in SVM for nonlinear data classification [194].

Table 3. 4: Kernel functions

Name	Expression
Linear kernel	$k(x_i , x_j) = x_i^{\mathrm{T}} x_j$
Polynomial kernel	$k(x_i , x_j) = (x_i^{\mathrm{T}} x_j)^d$ $d \geq 1$ denotes the degree of the polynomial when the degree reduced to $d = 1$ it represents the linear kernel.
Gaussian kernel	$k(x_i , x_j) = \exp\left(- \frac{\|x_i - x_j\|^2}{2\sigma} \right)$ also called RBF kernel, parameter $\sigma \geq 0$ controls the width of the gaussian.

3.6.6.4 Multiclass SVM

Since the SVM is originally designed for binary classification, thus it does not support multiclass classification natively. However, the binary SVM can be easily extended to accommodate multiclass classification. The following are the two commonly used approaches with the SVM to perform the multiclass classifications [111, 196]:

One versus One (OvO)

Given a training set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_i \in \{ C_1, C_2, \ldots , C_N\}$, based on the number of classes N, OvO approach constructs a binary classifier for each pair of classes resulting in $N(N - 1)/2$ classifiers. For example, if $N = 4$ classes, then we have $4(4 - 1)/2 = 6$ classifiers. OvO approach assigns a pair of classes Ci as positive and Cj as negative and trins the classifier

to distinguish between them. So, after classifying the new the test sample by all classifiers, the final class is determined by voting, and the class that received the most votes is assigned to the testing sample [194, 200].

One versus All (OvA)

This is also known as one versus rest, OvA approach constructs a binary classifier for each class, each time one of the classes is compared against the rest of classes n -1. For example, if N = 4 classes, then we have 4 classifiers and during the training each class is considered as a positive while the remaining 3 classes n - 1 are considered as negative. During the testing phase if one of the classifiers predicted a new test sample as positive, then this would be considered as the final classification. However, in case a number of classifiers have predicted the test sample as positive, then the final classification result is considered based on the prediction confidences, where the class with highest confidence is assigned to the testing sample [118, 194, 200]. A demonstration of OvO and OvA approaches are shown in figure 3.6.

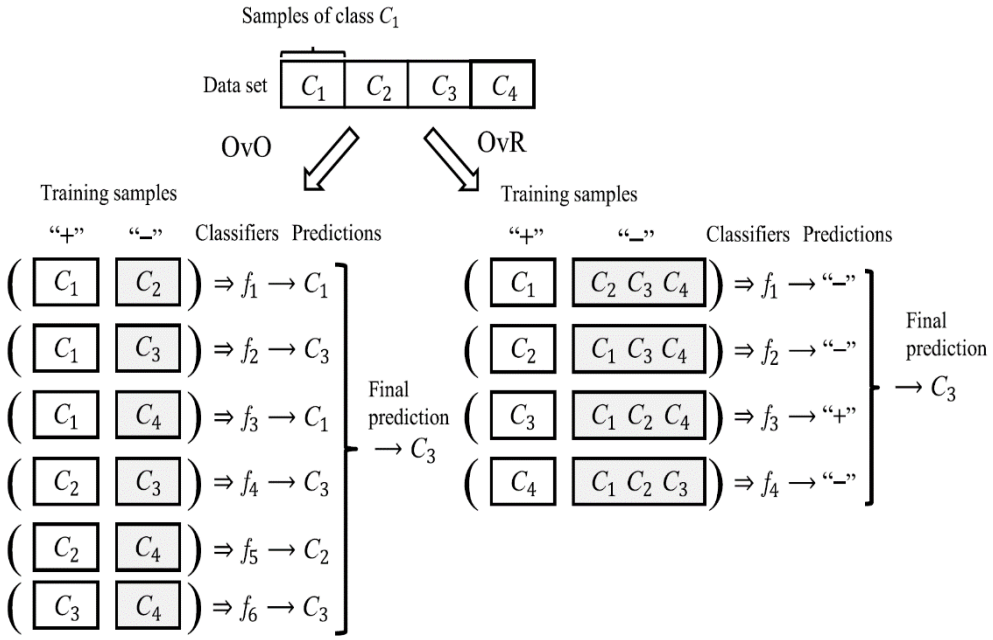


Figure 3. 6: Demonstration of OvO and OvA approaches [194]

3.6.6.5 Strengths and Weaknesses

Some common strengths and weaknesses of the SVM classifier are listed below[197, 201]:

Strengths of SVM

- Training the SVM can lead to a unique solution since is formulated as a quadratic programming problem, this is one of the key benefits of SVM over MLP, whereas an MLP classifier is known to have multiple local minima and thus may not be robust enough against the unseen data.
- Unlike the MLP which uses the sum of squares error to minimize the errors caused by outliers. As discussed in subsection 3.6.6.2, SVM uses the regularisation parameter C to control the misclassification error, by increasing the C value, a tighter margin is obtained, and more effort is made on minimising the number of misclassifications. Thus, outliers can be suppressed with a proper selection of the C value.
- Better performance than an MLP in the presence of limited data.
- One of the main differences compared to an MLP is mapping the nonlinear data from the original input space into a higher dimensional feature space, this provides an efficient and more accurate problem data analysis.

Weaknesses of SVM

- The selection of an appropriate kernel function is one of the main concerns of using the SVM. However, having selected an appropriate kernel function, more parameters need to be determined, including parameters of the selected kernel and the regularisation C parameter. Thus, determining the optimal values of these parameters is a time-consuming and challenging task.

- In the presence of large training data, SVM spends more time solving the dual optimisation problem due to a large number of training data and consequently a large number of Lagrange multipliers engaged in finding the support vectors.

3.6.7 Comparison of ML Classifiers

Sensitivity towards data modification

- Classifiers such as decision trees and MLPs fall into the category of unstable learners, the instability here means that a small modification in the training data that are used for training the unstable classifier leads to a significant change in the constructed classifier and therefore results in large differences in the prediction results.
- On the other hand, SVM, including other classifiers such as linear classifiers, Naive Bayes, and KNN fall under the category of stable learners. Since they are more robust against training data manipulation than unstable classifiers. Thus, small modifications in the training data do not significantly influence the performance of the stable classifier.
- Moreover, Random forest which was discussed in subsection 3.6.5, is an ensemble classifier based on the integration of multiple decision trees and formed by using a bagging technique along with a randomised selection of features. Since the significant feature of an unstable classifier is that a small modification in the training data results in a large change in the classification accuracy. Nevertheless, Bagging helps in reducing the variance of classification error, so that the classifier becomes more stable against data manipulation [173, 174, 194, 202].

Multiple classes

- Since SVM is primarily introduced for binary classification, thus in terms of multiclass classification, SVM and linear classifiers are incapable of handling multiple classes natively. However, the binary SVM can be extended to perform the multiclass classification using strategies such as OvO and OvA described in 3.6.3.4.
- In contrast, other classifiers such as KNN, MLP, Random forest, Naïve Bayes, and decision trees are naturally capable of handling the multiclass problem [203, 204, 205].

Model building (Training process)

- Since training the SVM requires solving the dual optimisation problem whose number of variables is equal to the number of training samples. Thus, in case of a large number of data used for training, SVM spends more time solving the dual problem due to both a large number of training data and the Lagrange multipliers involved in finding the support vectors.
- Furthermore, SVM model selection involves determining the SVM hyperparameters, this includes finding an appropriate kernel function which is one of the main concerns of using the SVM. Moreover, other SVM hyperparameters need to be determined such as parameters of the selected kernel and the regularisation C parameter. The process of finding the optimal values of these hyperparameters is called hyperparameter tuning which is time-consuming and not easy to perform.
- Similar to SVM the training process of MLP is slow since optimisation techniques are used during the training and repeated for several network weight initialisations in case a poor local optimum has been reached [177].

- Both SVM and MLP generate nonlinear decision boundaries, but with SVM the nonlinear boundary is based on a user-defined kernel function, whereas for MLP the nonlinear boundary is learned and refined during the training process which might require more data and training time. Moreover, Similar to SVM determining the optimal hyperparameters of an MLP i.e., number of hidden layers, number of neurons in each layer, activation functions, etc. is difficult and time-consuming [197, 201, 206].
- Moreover, decision tree and Random forest can be constructed without the need for data normalisation or scaling. However, compared to a single tree classifier, Random forest requires more time to build since a large number of trees are used as base learners [207].
- Finally, since KNN is a representative of lazy learning so unlike other classifiers it has no explicit training phase it simply stores the entire training set during the training phase. In the testing phase for a given testing sample, it searches through the training set for the k nearest instances based on distance metric and classifies the testing sample according to the majority class among k neighbours. Thus, all the computational requirements are delayed till the classification is performed [194].

Interpretability of classification models

- SVM and MLP are criticised as black box classifiers since the learned classification models cannot be explicitly understood or interpreted by users. The comparison of black-box classifiers is often made with white box classifiers whose produced classification models provide more transparent outputs that are easier for users to interpret [111, 194].
- White box classification models include decision trees, Bayesian network classifiers, and KNN. On the other hand, examples of black

box classification models include SVM, MLP, and ensembles of classifiers [208, 209].

- Furthermore, the term grey box could be used to describe some ensemble-based classification models whose base learners of white boxes. An example of a grey box model is a Random forest, which is generally not a white box model since a large number of trees are used as base learners in which each tree is heavily influenced by random selections of samples and features. Hence it is not directly interpretable by users. However, the Random forest model is still partially interpretable by measuring the importance of features in all trees in the forest and then sorting the features in descending order according to their importance [208].

Sensitivity towards outliers

- MLP classifier is vulnerable to outliers since it uses the sum of squares error to minimize the errors posed by outliers. Whereas in SVM the regularisation parameter C is used to control the misclassification error, so in case the value of C is large then a tighter margin is obtained and misclassification is suppressed. Thus, by a proper selection of the C value, outliers can be suppressed [201].
- KNN is also sensitive to outliers, since KNN classifies the new test record by searching through the training set for the most similar or nearest instances in the region of k-neighbourhood based on the value of k, thus in case of existing outliers in the region of k-neighbourhood. The classification performance of KNN is heavily affected by the outliers, If the value of k is small then the classification decision is heavily influenced by outliers, while a larger value of k might have more outliers [210].
- Moreover, the performance of a single decision tree classifier could be affected by the presence of outliers, whereas Random forest is more

robust to outliers, it improves the resilience of a single decision tree against outliers because of the randomness it provides with respect to samples and features [211, 212].

Types of data

- Many ML classifiers including but not limited to SVM, MLP and KNN are incapable of dealing with categorical data and they require that all their inputs must be numerical. Thus, categorical features must be transformed into numerical values using encoding techniques such as label encoding and one-hot encoding.
- On the other hand, classifiers such as decision trees, Random forest and Naïve Bayes can naturally deal with categorical and numerical features without a need for encoding [213, 214, 215].

High dimensional data

- In the presence of high dimensional data, i.e., when a large number of features exist in a dataset, issues such as data sparsity and the difficulty of distance calculation are faced by ML classifiers. This fact is known as the curse of dimensionality. So, with a large number of features, KNN is more sensitive to the curse of dimensionality and performs much worse than other classifiers. This is because, in a high dimensional space, similar data points will have a large distance between them caused by the increase of dimensionality.
- Conversely, compared to other classifiers SVM is more effective in dealing with high dimensional data [193, 214, 216].

3.7 Overview of Other Deep Learning Methods

DL is a subset of ML, which is constructed using multiple layers of ANNs. Compared to simple ANNs, DL models have more hidden layers that are organised in deeply nested network architecture [119, 120, 124].

But what particularly makes DL algorithms unique and different from ML ones is their ability to automate the process of feature extraction, this allows researchers to extract discriminative features with minimal human effort and domain knowledge [119].

DL algorithms follow the same mechanism of employing multiple hidden layers wherein the information is processed layer by layer, thus through the multilayer architecture, the original input representation is gradually converted from low-level feature representations into high-level feature representations. For this reason, DL is also referred to as feature or representation learning [119, 194]. Figure 3.7 shows the optimised features are learned in an automated way [125].

In recent years, there has been a rapid increase in the use of DL, this is due to the availability of large datasets and advances made in hardware technologies in particular the computational power of Graphics Processing Unit (GPU) [119]. Some of the most common applications of deep learning include Natural Language processing (NLP), computer vision, pattern recognition, signal processing and anomaly detection. Whereas the commonly used DL algorithms include Convolutional Neural Networks (CNNs) [121], Auto Encoders (AE) [122], Recurrent Neural Networks (RNNs) [123] and Deep Belief Networks (DBNs) [124].

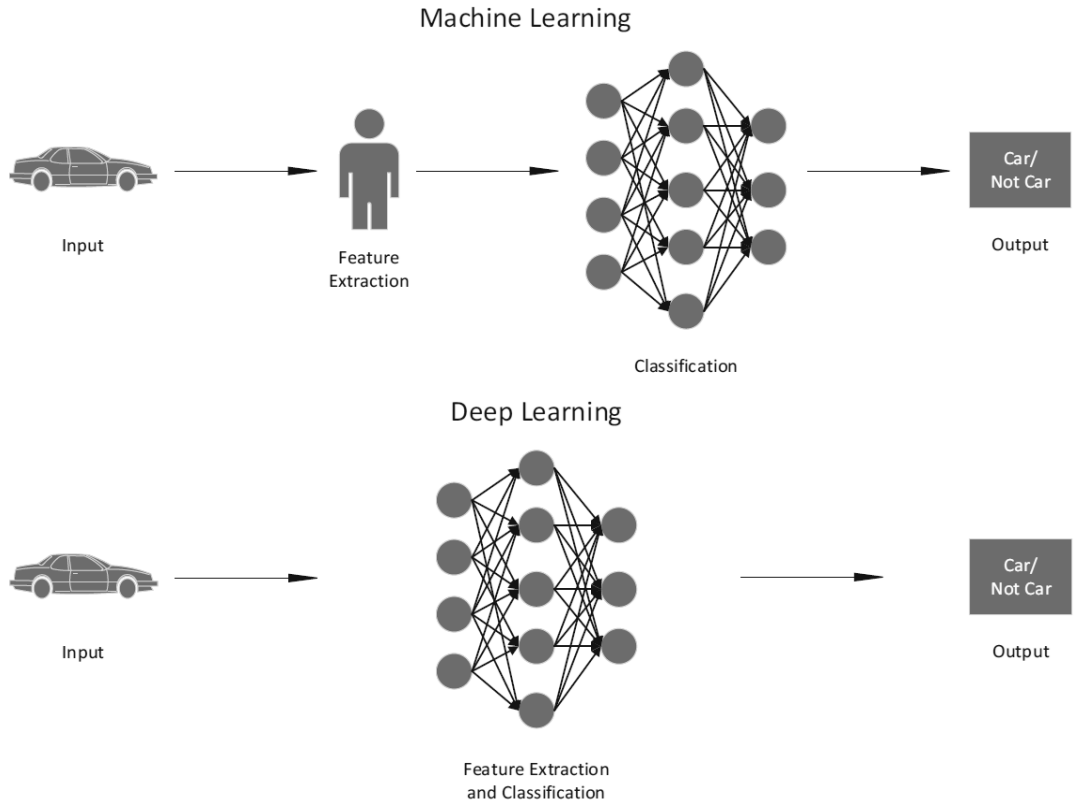


Figure 3. 7: Automatic feature extraction [125]

Furthermore, it is important to understand when to apply DL instead of ML, for example, it will be challenging to use ML algorithms such as SVM or Random forest in certain domains such as computer vision and NLP since tasks related to these domains such as detecting and classifying objects within images or understanding and translating language are relatively easier to perform using DL algorithms.

However, in cases where the application of the ML algorithm would suffice, then it should be considered rather than the DL because; computationally is less expensive, the constructed model is more interpretable by users and finally, it requires less data compared to DL algorithm [120, 125].

DL is known to be data-hungry and computationally expensive, training DL algorithm from scratch requires a sizeable amount of data and intensive

computational power in order to achieve a well-behaved performance model [119, 125]. However, in cases where there is a shortage of data the concept of Transfer Learning (TL) can be utilised to tackle the issue of undersized training data.

TL is a technique that allows transferring the knowledge of a model that previously has learned on a source dataset to a target dataset [219]. For example, training the CNN on a sizeable amount of data, during the training process bias and weights are also learned, these parameters are then transferred to a similar new model with less data and used for retraining the new model instead of starting the process of training from scratch [119].

Fine-tuning is a common TL technique described in the following steps [219]:

- Train a new ANN model, this is a source model trained on a source dataset.
- Create a new ANN model, this would be the target model which copies the source model's design and its parameters except for the output layer, with the assumption that these parameters hold the knowledge learned from the source dataset which also can be applied to the target dataset.
- Next, add the output layer into the target model, whose outputs' number corresponds to the classes in the target dataset.
- Finally, train the target model on the target dataset, only the parameters of the output layer are trained from the scratch, while parameters of all other layers are fine-tuned based on the target dataset.

3.8 Review of ML Methods for Network Traffic Classification

ML based classification methods are most suitable for the classification of network traffic. These methods can differentiate different smartphones' applications based upon the traffic generated by them.

Successful implementation of ML methods depends on the features extracted from network traffic to discriminate their applications. These features are generally extracted from network packets and payload size [126].

Several ML methods have been successfully implemented for network traffic classification. For instance, Shbair et al. [127] proposed a two-tier hierarchical framework for network traffic classification. Their proposed framework can recognise different kinds of services running within the Hypertext Transfer Protocol Secure (HTTPS) connection based upon payload size and the time interval between constructive packets. They used decision tree and Random forest classification methods. The authors reported recall of more than 95% based upon real internet traffic.

Aceto et al. [128] designed a hybrid classification method for differentiating encrypted mobile traffic by combining multiple classification methods. The authors reported improved performance of their proposed method over state-of-the-art ML methods by 9.5% in terms of recall.

Fu et al. [129] proposed a method named as CUMMA to classify services in mobile messaging applications. The authors employed Random forest, hidden Markov model and clustering methods for detecting different services in mobile messaging applications. They reported an accuracy of more than 96% for WhatsApp and WeChat based traffic generated from 15 volunteers.

Taylor et al. [130] proposed a scanner application framework called AppScanner. Their framework helps to fingerprint and identify mobile applications. The authors demonstrated the performance of their framework based upon network traffic generated by 110 most popular apps available at Google Play Store. They pre-processed the network traces for removing background traffic and extracting features of network packets. They used a SVM and Random forest classification methods and reported more than 99%

accuracy for recognising different applications using their framework. Authors of [130] have extended the AppScanner framework in [134] along several dimensions.

Alan and Kaur [131] employed supervised ML methods for classification of android apps. They used launch time traffic of TCP/IP headers to distinguish android apps. They captured network traffic of 86,109 app launches by repeatedly running 1,595 apps based on four Android devices. They concluded that the first 64 packets in network traffic can be identified with an accuracy of 88%, when the ML learning methods are trained and tested on the same device. They also demonstrated that when the data from another device (operates on different operating system) is used for testing, an accuracy of only 67% could be achieved.

The authors of [10] used SVM to classify the network traffic for a number of smartphone applications into two categories. They applied a 6-fold cross validation mechanism and achieved an accuracy of 88.1% for the SVM classification model.

Zhao *et al*. [132] proposed RobotDroid, which is based on an SVM algorithm. The proposed framework detects unknown malware attacks on smartphones and mainly focuses on the disclosure of the confidential information, like private information of the users, payment/sales related information, etc.

On the other hand, Stöber *et al.* [133] suggest a scheme in order to identify the network traffic by utilising the characteristics of the traffic patterns coming from the devices. The research also suggested that 70 percent of the traffic belongs to the activities that are running in the background; hence, creating fingerprints by using those activities. By creating those fingerprints, the model can compare the incoming traffic from the fingerprints and then identify the network traffic.

Erman et al. [135] performed a comprehensive experiment for comparing the performance of semi-supervised ML methods for network traffic classification. The authors aim to evaluate the performance power of ML methods for detecting new application.

The authors in study [136] have evaluated clustering methods, DBSCAN and K-means clustering methods for network traffic classification. The results demonstrated that both K-means clustering and DBSCAN clustering methods exhibited better performance in network traffic classification. DBSCAN clustering method provided lower accuracy in comparison to K-means clustering method, but it produces better clusters.

Williams et al. [137] also performed a comprehensive comparison of five ML methods for internet traffic classification. The authors reported that the decision tree achieved the maximum accuracy in internet traffic classification.

Bernaille et al. [138] focused on the K-means clustering method for classifying network flows based on the first five network packets in the network flow. They aimed to classify the real-time network traffic using the clustering method.

Several researchers have employed deep learning methods for classifying different applications based upon traffic generated by them.

For instance, Aceto et al. [126] use deep learning methods for classifying mobile encrypted traffic based upon three data sets of real human user activity. They critically analysed the use of deep learning methods in classifying mobile encrypted traffic.

Chen et al. [139] applied deep learning for classifying malware traffic using features extracted from raw data and handcrafted features. They used the weighted back propagation method and hierarchical learning to handle the

imbalanced dataset issue for classifying malware traffic. The authors reported 99.63 % accuracy and 85.44 % for precision based upon a synthetic dataset.

Wang et al. [140] proposed a method for malware classification based on 2D-CNNs, using two different choices of raw traffic images. They evaluated the performance of their method using a self-generated dataset (of ≈ 752k instances). Which is organised into two parts: ten types of malware traffic from public websites and ten types of normal traffic. They employed 2D-CNN for two different scenarios: malware / normal (binary) classification and traffic type classification (20 classes).

3.9 Summary

In summary, this chapter reviewed different network traffic classification approaches employed in communication networks, it also reviewed the methods of ML based network traffic classification. This was followed by an explanation of the necessary steps required to perform ML based traffic classification. An ML classifier taxonomy based on different properties was also presented in this chapter, including a description of which ML classifier goes with which kind of data. The chapter also described the commonly used ML classifiers that were adopted in this research, including a comparison of these ML classifiers. An overview of other DL methods was also provided in this chapter. This was followed by a review of ML methods proposed in the scientific literature for network traffic classification.

4

OPTIMISING WLANS POWER SAVING

4.1 Introduction

This chapter introduces the Context-aware Listen Interval (CALI) approach for optimising WLANs power saving. Section 4.2 describes how smartphone applications' network traffic reflecting a diverse array of network behaviour and interaction were exploited to provide the contextual inputs for training ML classifiers of the output traffic, thus building an ML classification model. Section 4.3 begins by justifying the selection of the chosen applications and the assignment of output modes. It then discusses how CALI power saving modes were employed to optimise the sleep and awake cycles of the WNIC based on the classified output traffic. This is followed by describing the process of data extraction and preparation employed in this research for constructing the dataset (section 4.4). Section 4.5 begins by describing the experimental settings employed in this chapter for traffic classification, including the description of parameter settings for the selected ML classification models. It then evaluates the performance of ML classifiers on Dataset 1 using 10-fold cross-validation prior to applying feature selection methods. It also evaluates the performance of selected ML classifiers after applying feature selection methods on Dataset 2CBFS and Dataset 3IGFS, both using 10-fold cross-validation.

4.2 Context-Aware Listen Interval (CALI)

In ML, classification is defined as a learning method that maps or classifies instances to corresponding class labels which were predetermined in a given

dataset. According to Han et al. [141] data classification is a two-step process; the first step is learning, where a classification model is built from a given dataset, the data from which a classification model is learned called a training set. The second step is classification, where a model is used to predict class labels for previously unseen data. The dataset, which is used to test the classifying accuracy of the learned model is called the test set.

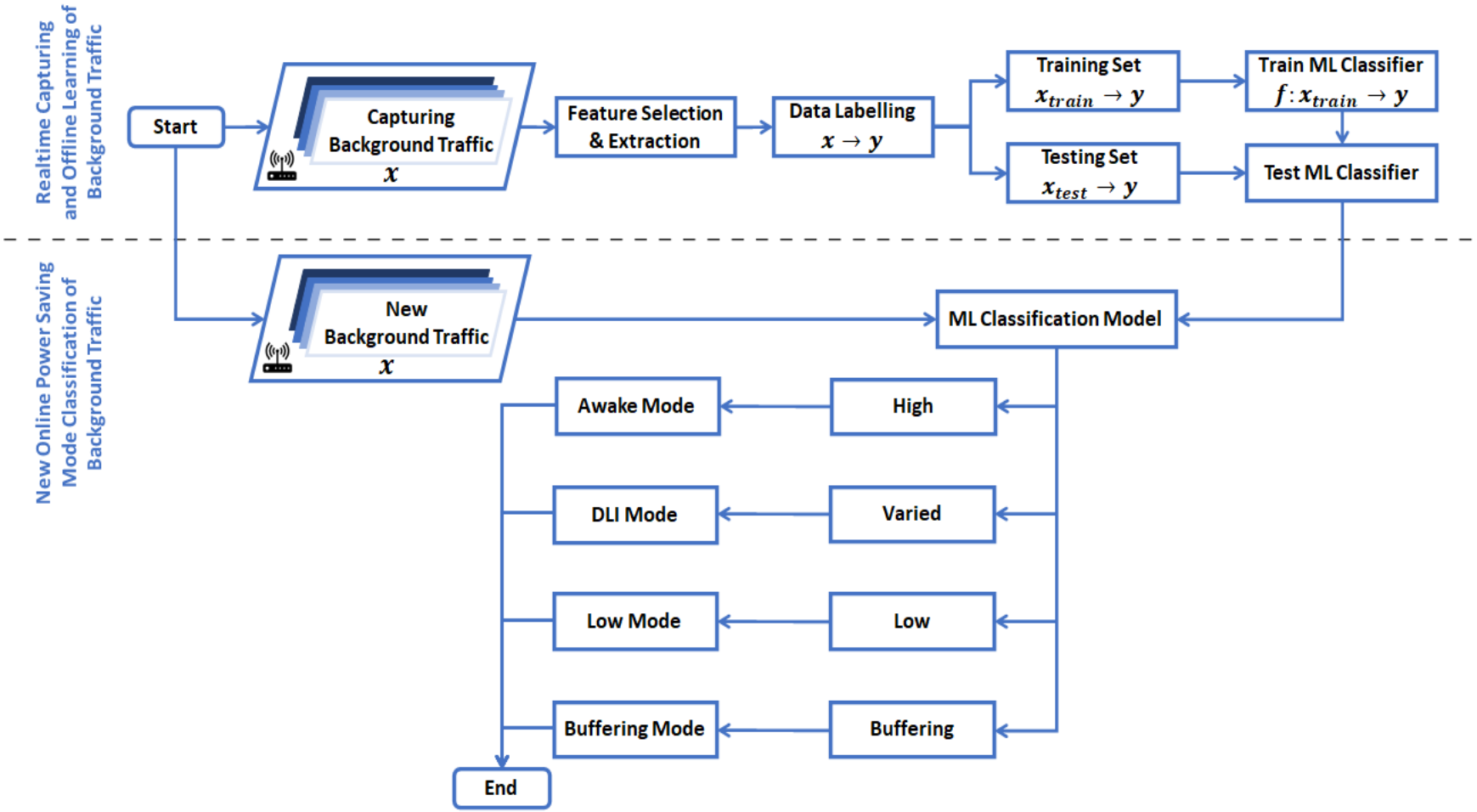
In [57] we have constructed a real-world dataset based on the network traffic of nine smartphone applications, which reflects a diverse array of network behaviour and interaction. For high levels of network interactivity, both Google Hangouts and Skype audio and video calls were selected. For traffic with intermittent interactions, Gmail and Facebook were chosen. For applications with the lowest level of interactions New Star Soccer (NSS) and New Star Cricket (NSC) were considered. Network interactions of these applications mostly occur during fetching advertisements. For the network traffic that reflects applications with audio buffering capabilities, the traffic of XiiaLive internet radio application was captured.

Figure 4.1 shows the flowchart of CALI, where instances of real-time network traffic of each application were captured, and then manually labelled to the right output or class. We have labelled all instances of applications with a high level of interaction as high, instances of applications with an intermittent level of interaction were labelled as varied, whereas instances of applications with the lowest level of interaction were labelled as low. Finally, instances of audio streaming application with buffering capability were labelled to buffering.

After labelling the input samples of the captured traffic of each application, an ML classifier learns to map the input features of each sample to an output class from the training set, constructing the ML classification model.

The next step is the classification, where the ML classification model is used to predict class labels for previously unseen data. Test set is used to test the classifying accuracy of the learned model. Subsequently the ML classification model is capable to identify the new unobserved samples of traffic as one of the pre-defined classes, high, varied low and buffering, e.g., the ML classification model assigns samples of highly interactive application to the class high in accordance with the training accomplished in the previous step.

Figure 4. 1: Context-aware listen interval



4.3 CALI Power Saving Modes

It is important to mention that the selection of the nine applications used in this research was initially driven by [10, 41] where authors of these studies described the network interaction of a wider range of smartphone applications, these include: 1- real-time applications with high and constant levels of network interaction. 2- Applications with intermittent levels of interaction that do not always receive data, as they run in the background while the screen is off and the WNIC wakes up intermittently when downloading content. Moreover, 3- applications with the least levels of network interactivity, these applications mostly are offline except for periodically fetching advertisements. Finally, 4- audio streaming applications with playback buffering capacity, where these applications are able to buffer the audio stream.

Therefore, the criterion for selecting the nine applications employed in the research was based on the four types of variation in the levels of network interactions of the applications described in [10, 41].

Moreover, the authors in [10] assigned only two sets of priorities or modes, high and low, for the described four different types of network traffic of smartphone applications. Wherein no further priority levels or modes were assigned by them to cater for firstly, applications with intermittent levels of interaction, secondly, applications with the least levels of network interactivity, and finally audio streaming applications with playback buffering capacity. Instead, they assigned the three types of applications' network into a single mode, low.

So, having four modes assigned intuitively according to the described four types of network traffic would be more efficient. Therefore, in CALI, four modes: 1- Awake, 2- DLI , 3- Low and 4- buffering were assigned to cater for

the four different types of network traffic of applications, and the experimental results in chapter seven, subsection 7.2.2 and 7.2.3 showed their efficacy in optimising the sleep and awake cycles of the WNIC and reducing energy consumption.

Moreover, in this research, we focused on the network activity of a single smartphone application opened at a given time, therefore no additional output class was assigned to cater for the network traffic of applications running simultaneously such as an audio streaming app running in the background while using a lower network usage gaming app.

Furthermore, there is room for further investigation to find the optimal number of power saving modes that could be associated with the CALI's framework. This would be based on further analysis of the captured network traffic of the applications and based on the analysed network traffic, optimal number of power saving modes can be determined and incorporated into CALI's framework.

However, in order to optimise the sleep and awake cycles of the WNIC in accordance with the applications' network activity, we have defined four CALI power saving modes.

These power saving modes enable additional power saving opportunities and have been devised based on the classified output traffic of the captured samples from a varied range of smartphone applications' network traffic that reflect a diverse array of network behaviour and interactions. Hence, the ML classification model classifies the new unseen samples into one of the output classes, and the WNIC will be adjusted to operate into one of CALI power saving modes.

Moreover, CALI handles applications, which it cannot map to one of the four modes by reverting the WNIC to operate in SPSM mode. That means, the

worst possible performance is that of SPSM, but if one of the four modes applies, a significant performance improvement with respect to power saving is achieved.

- **Awake Mode**

When the ML classification model classifies the new unseen samples of highly interactive applications to the output class high. Consequently, the WNIC is set to operate in awake mode.

- **DLI Mode**

The ML classification model classifies the traffic samples of applications with varied levels of interactivity to the output class varied. The WNIC will be adjusted to operate in DLI mode. We have considered employing the DLI methodology introduced in [18]. So, the listen interval is incremented by 1 at each time a wireless device wakes up during the listen interval and finds no packets buffered at the AP. The listen interval reverts back to 1 when interactions occur. To prevent the listen interval from growing excessively we set an upper bound of 10 = 1000ms for the listen interval. Applications such as Gmail and Facebook have intermittent network interactions and do not always receive data. Therefore, assigning the background traffic of these applications to the awake mode would not be efficient.

- **Low Mode**

The ML classification model classifies the traffic of applications with the lowest level of interactions to the output class low. Consequently, the WNIC will be switched to operate on low mode, with an extended value of the listen interval. This is beneficial as network interactions of these

applications e.g., NSS and NCS mostly occur during fetching advertisements.

- **Buffering Mode**

The ML classification model classifies samples of audio streaming applications with buffering capability to the output class buffering. The WNIC will be set to operate in buffering mode. The buffering mode was defined for applications that allow users to stream audio over the Internet, according to [142] these applications are capable to buffer several seconds of audio stream. For such applications, switching off the WINC for short periods of time does not impact on the playback streaming quality.

4.4 Data Extraction and Preparation

This section describes the process of data extraction and preparation employed in this thesis. We have constructed a dataset by capturing real-time network traffic of nine selected smartphone applications depicting different types of network behaviour and interactions. Table 4.1 shows the chosen applications and the degree of network interactivity of each application. All the applications including NetworkLog installed from the official Google play store. 150 instances of the network traffic of each application were captured with the aid of NetworkLog, and the running time of 25 minutes resulted in a total of 1350 instances. Samsung Galaxy S5 is used to capture all the instances of the background traffic of the entire applications running Android version 6.0.1.

Table 4. 1: Applications and the degree of network interactivity

Applications	Degree of interactivity
1- Skype video call 2- Google Hangouts video call 3- Skype voice call 4- Google Hangouts Voice call	High level of interactivity
5- Facebook 6- Gmail	Varied level of interactivity
7- New Star Soccer (NSS) 8- New Star Cricket (NSC)	Low level of interactivity
9- XiiaLive internet radio app	Buffer stream

The 9 applications represent different types of network behaviour and interactions, for high level of network interaction; we have considered video and voice calls of Skype and Google Hangouts. For the varied level of interactions Facebook and Gmail have been chosen, for Gmail, 23 emails were received at random instances. And 23 tagged posts were received at random instances for Facebook as updates. NSS and NSC were chosen to represent all applications with a lower degree of interaction, these applications mostly are offline and the interaction mostly occurs during fetching advertisements. Finally, to represent applications with audio buffering capability the XiiaLive internet radio application was considered; we chose a random station streaming at 128 kbps.

We have manually labelled instances of the nine applications according to the levels of traffic interactivity in the background of each application. Figure 4.2 shows the receiving data rate in Kbytes/sec of the first 50 instances which reflect varying levels of network interaction.

So, 1350 instances are used in the construction of a dataset, named Dataset 1, with 150 instances per application and 6 features per instance. Furthermore, four output classes were assigned to cater for the network traffic of these applications.

Thereby out of the nine chosen applications, the first output class was assigned to the four applications that represent real-time applications with high and constant levels of network interaction. The reason for having four applications for this output class is to ensure more variation in the range of network traffic included in the training data by having two VoIP applications and two video-calling applications. These applications are: 1-Skype video call, 2- Google Hangouts video call, 3- Skype voice call, and 4- Google Hangouts Voice call. Consist in a total of 600 samples and were assigned to class high.

For the remaining three types of network traffic, the second output class was assigned to the two applications that represent network traffic with intermittent levels of interaction. These applications are: 5- Facebook, and 6- Gmail. consist in a total of 300 samples and were assigned to class varied.

While the third output class was assigned to the two applications that represent the least levels of network interaction. These applications are: 7- New Star Soccer (NSS), and 8- New Star Cricket (NSC) consist of a total of 300 samples and were assigned to class low.

Finally, the fourth output class was assigned to one application that represents the network traffic of audio streaming applications. This application is: 9- XiiaLive internet radio app. Consists of 150 samples and was assigned to the class buffer.

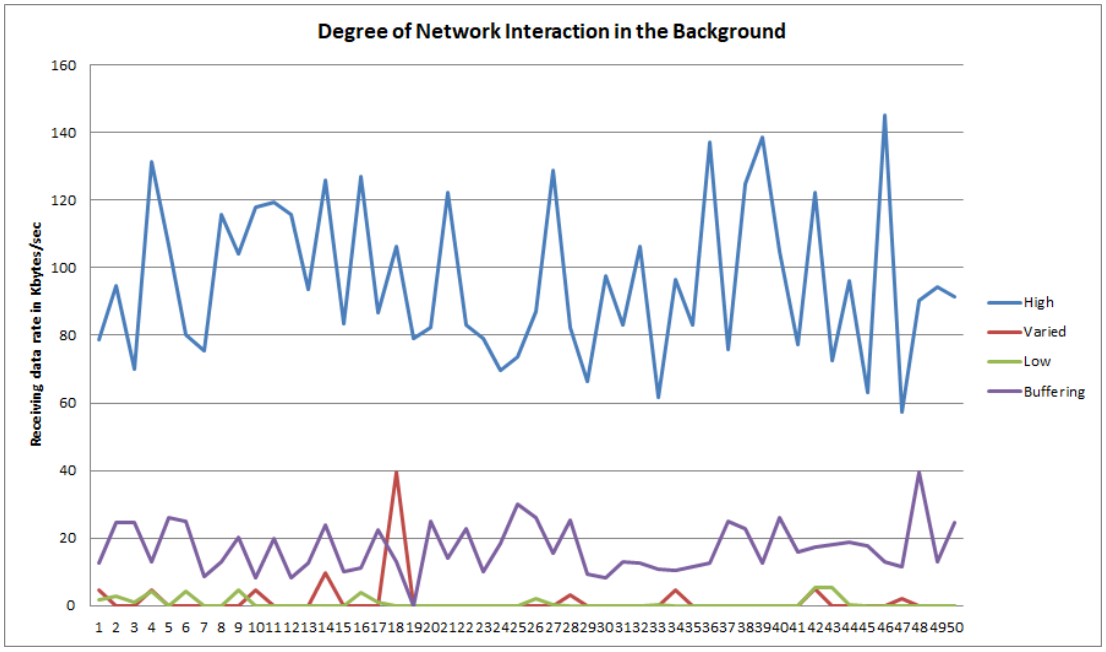


Figure 4. 2: Arrays of network behaviour characterised by levels of traffic interaction

Table 4.2 shows the full set of features extracted using NetworkLog from the background of each application.

Table 4. 2: Full set of 6 features

Feature Number	Feature Name
1	Receiving data rate in Kbytes/sec
2	Transmitting data rate in Kbytes/sec
3	Total received Kbytes
4	Total Transmitted Kbytes
5	Total number of received packets
6	Total number of transmitted packets

These features are statistical-based and unique for specific types of applications. Additionally, inspection into the packet content is not required to extract these features, hence statistical features have low computational overhead and are applicable for both encrypted and unencrypted traffic [60,

63]. Moreover, these features reflect the applications' network interactivity better than non-network features like touch screen rate, as regularly touching the screen, does not always mean that network traffic is occurring. For instance, video games are highly interactive in terms of user and screen, but practically non-interactive in terms of network interaction.

4.5 Initial Experiments (Traffic Classification)

This section describes the experimental setup employed in this chapter for traffic classification, this is followed by performance analysis of the five ML classifiers on Dataset 1, Dataset 2CBFS and Dataset 3IGFS using 10-fold cross validation, in terms of classification accuracy, precision, recall and f-measure.

4.5.1 Experimental Setup

The main purpose of feature selection is to minimise the set of features by eliminating any irrelevant and redundant features resulting in less computational complexity, higher classication accuracy and maximised generalization capability [143, 144]. Moreover, in terms of energy consumption, the fewer the features the better they are for smartphone energy saving.

Since the wrapper-based feature selection methods involve learning algorithm in the elimination of irrelevant and redundant features, they are slow and computationally expensive [145]. Therefore, to extract eliminated versions of datasets, two widely used filter-based feature selection algorithms were considered. These are Consistency Based Feature Selection (CBFS) and Information Gain Feature Selection (IGFS).

CBFS evaluates all the subsets of features, in order to determine the smallest optimum subset of features, which is consistently capable to map to the output class as with full set of features. whereas IGFS evaluates all the features with

the output class; features with higher information gain value to the output class are selected. The best first search method was applied to attribute selection for CBFS algorithm, while the ranker method was selected for the IGFS algorithm.

The WEKA tool was used, for the extraction and the application of the reduced features' datasets, named as Dataset 2CBFS and Dataset 3IGFS. 1350 data samples were included in each set representing the total of 9 applications, the total number of six features included in Dataset 1, reduced number of 4 features were included in Dataset 2CBFS, and Dataset 3IGFS. Table 4.3 shows the list of features after applying the CBFS algorithm, while Table 4.4 shows the list of features after applying the IGFS algorithm, the top 4 features in ranking were chosen.

Table 4. 3: Set of features for Dataset 2CBFS

Feature Number	Feature Name
1	Receiving data rate in Kbytes/sec
2	Transmitting data rate in Kbytes/sec
3	Total received Kbytes
4	Total Transmitted packets

Table 4. 4: Set of features for Dataset 3IGFS

Feature Number	Feature Name
1	Total received Kbytes
2	Total Transmitted Kbytes
3	Total number of received packets
4	Total number of transmitted packets

The experiments were performed with the aid of WEKA [146], a well-known ML tool, applied in many studies including [147, 148]. v3.6.12 on a desktop

computer operating Microsoft Windows 7 Enterprise with Intel core i7-4770 CPU of 3.40 GHz and 4 GB of RAM, located within the university campus.

For validating the accuracy of the ML classifier in predicting/mapping the inputs to the correct output class, and based on the recommendation of [149], cross validation of K =10 is used to avoid over-fitting and to see how well ML classifiers perform in classifying the unseen samples. Thus, the dataset is divided into 10 N equal parts or portions, each portion (1/N) is used for testing, while the remaining ((N − 1)/N) are used for training.

In order to determine the suitable ML classifier in terms of classifying smartphone applications' network traffic based on different levels of behaviour and interaction, the performance of the 5 ML classifiers described in section 3.6 will be analysed. Thus, the chosen ML classifiers are MLP, KNN, SVM, decision tree (C4.5), and Random forest.

The performance of each classifier evaluated is based on the following metrics:

Classification accuracy: is an evaluation metric that estimates the overall correctness of model's predictions. It calculated by dividing the number of correctly classified instances by the total number of instances:

Accuracy = (Correctly Classified Instances) / (Total Instances).

Precision: is an evaluation metric that measures the proportion of correctly classified instances of a particular class out of all the instances that the model classified as that class:

Precision = True Positives / (True Positives + False Positives).

Recall: also known as sensitivity, is an evaluation metric that estimates model's ability to correctly identify the positive instances out of all the actual positive instances.

Recall = True Positives / (True Positives + False Negatives).

F-measure: also known as the F1 score, this evaluation metric combines both precision and recall into a single value, is defined as the harmonic mean between precision and recall and is calculated as follows:

F-measure = 2 * (precision * recall) / (precision + recall).

Moreover, it is a common practice to initially train the ML model using the default hyperparameter setting as the baseline model and subsequently conduct a hyperparameter optimization process to enhance the model's performance [239].

Therefore, in this thesis, we have adopted this approach. Firstly, we trained the selected ML models using the default hyperparameter settings listed in table 4.5 as baseline models for the following experiments and experiments conducted in sections 5.2 to 5.5. We then conducted a hyperparameter optimisation process in sections 6.4 to 6.10 to further enhance the model's performance. This was similarly followed in [240, 241, 242] where the authors of these studies initially trained the ML model using the default hyperparameter settings suggested by the WEKA tool and then conducted a hyperparameter optimisation process.

Table 4. 5: WEKA default parameter settings

Classifier	Parameter values
MLP	- The model is defined in 3 layers (an input layer, a hidden layer, and an output layer). - Number of nodes in an input layer: 6 nodes correspond to 6 features for Dataset 1. Whereas the number of nodes would be 4 for Dataset 2CBFS and Dataset 3IGFS. - Number of nodes in hidden layer: - H = "a" = 5 nodes in a single hidden layer. In WEKA the parameter (-H) represents a number of hidden layers and the number of nodes in each layer, where the default setting of this parameter in WEKA is "a" which creates a

	network with a single hidden layer and the number of nodes = (number of features + number of classes) /2. Thus, in the case of Dataset 1 with 6 features, the number of nodes would be 6 + 4 / 2 = 5 nodes in a single hidden layer. Whereas the number of nodes would be 4 for Dataset 2CBFS and Dataset 3IGFS. - Number of nodes in an output layer: 4 nodes correspond to 4 classes. - Learning rate (-L) = 0.3. - momentum (-M) = 0.2. - Activation function for nodes in the network = sigmoid. - Number of epochs (-N) to train through = 500. - Batch size = 100. - The value of seed (-S) is used to seed the random number generator which affects setting the initial weights of the connections between nodes. However, by default the value of this parameter is = 0. - Normalising the attributes is turned on by default. - Validation Set Size (-V) by default is set to 0, meaning no percentage of the data being set aside for validation and instead the network will train till the specified number of epochs is reached. - Occurrence of decaying (-D) the learning rate, this parameter is disabled by default.
KNN	- Number of nearest neighbours (-K) used: by default, = 1. - Distance function: Euclidean distance. - Attribute Indices: comma separated list of attribute indices with first and last valid values. - Normalisation of the attributes: this is turned on by default. - Distance weighting: by default, no distance weight assigned. - Nearest neighbour search algorithm (-A): a linear search by default is used. - Window Size (-W): it restricts the number of training instances maintained; it drops old instances above the value being specified according to FIFO. The default value of this parameter is = 0 means no restriction to the number of training instances. - Cross Validate (-X): For training data, if set to true then it determines the optimal K value between 1 and specified K value by using hold-one-out cross-validation. By default, this parameter is set to false. - Number of decimal places to be used for the output of numbers in the model: = 2. - Batch Size: 100.
SVM	WEKA SVM library: SMO. Multiclass classification problems are solved using pairwise OvO strategy.

	- Regularisation parameter (-C): the default value of this parameter = 1.0.
	- Type of kernel (-K): normalised polynomial kernel.
	  - Exponent (-E) value or degree of the normalised polynomial kernel: by default, E = 1.0.
	  - Cache size: the size of the cache for the normalised polynomial kernel by default = 250007.
	- Calibrator: Logistic Regression.
	  - Ridge value (-R) = 1.0E-8.
	  - Maximum number of iterations to perform (-M): default = -1, means until convergence.
	  - Number of decimal places to be used for the output of numbers in the model: = 4.
	  - Batch Size: 100.
	- Normalisation of the attributes: this is turned on by default.
	- Tolerance parameter (-L): which determines the stopping criterion for optimization process. By default, is = 0.001.
	- Number of decimal places to be used for the output of numbers in the model: = 2.
	- Epsilon parameter (-P): which is used for controlling the round-off error. By default, is = 1.0E-12.
	- Random Seed (-W): Random number seed for the cross-validation, by default = 1.
	- Number of folds (-V): for cross-validation used to generate training data for calibration models is -1 means use training data.
	- Batch Size: 100.

Decision tree (C4.5)	Criterion: uses the gain ratio criterion by default to select the best split. - Confidence factor (-C): that controls the pruning of the tree = 0.25. - Minimum number of instances (minNumObj or -M) that must be present in a leaf node =2. - Reduced error pruning (-R): this parameter specifies whether to use reduced error pruning instead of standard C4.5 pruning, however, is not enabled by default. So = False - Number of folds (-N): when reduced error pruning is selected, this parameter specifies the number of folds used for reduced error pruning where data is divided equally into specified parts and the last one used for pruning. However, the default value is 3. - Unpruned (-U) : is turned off by default, if enabled it builds unpruned tree. So pruning is performed by default. - Subtree raising (-S): Whether to perform the subtree raising pruning operation. This parameter is enabled by default.

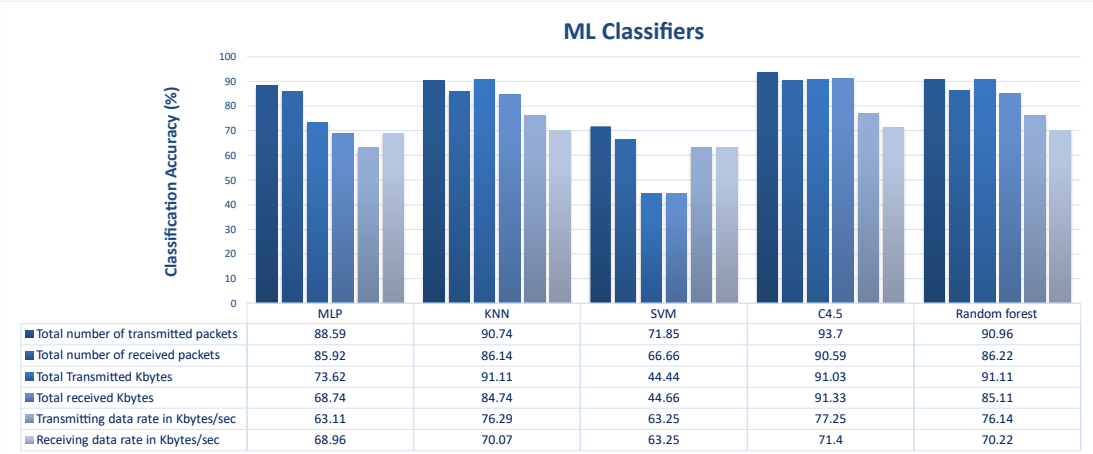
	- Number of decimal places to be used for the output of numbers in the model: = 2. seed (-Q) for randomising the data when reduced-error pruning is used then its default value is = 1. - Do not make split point actual value: set to false by default. However, If true, the split point is not relocated to an actual data value. - Use minimum description length correction: when finding splits, by default is set to true. - Collapse tree: is set to true, so parts are removed that do not reduce training error. - Bath size = 100.
Random forest	- Number of trees (-I) = 100. - Maximum depth (-depth): of trees indicates how deep the tree would be, by default is set to 0 for unlimited depth. - Number of features to consider in each split point (-K): is 0 which = log2(number of features). - Seed (-S) for random number generator: the default value of this parameter is = 1. - Bag size percent (-P): this parameter determines the percentage of the training data that is used for building each tree. By default, the value of this parameter is = 100. - Minimum number of instances (-M): minimum number of instances that must be present in a leaf node, by default is = 1. - Compute attribute importance: this parameter is used for computing and outputting feature importance based on average impurity decrease, is set off by default. - Break ties randomly: it breaks a tie randomly when multiple features have the same importance. However, is set to off by default. - Number of execution slots: use for constructing the ensemble by default, is = 1. - Minimum variance for split (-V): default = 1e-3. - Batch size: 100.

Unless stated otherwise the parameters in table 4.5 are used in the subsequent experiments.

4.5.2 Results and Analysis

This section analyses the performance of the five ML classifiers on Dataset 1, Dataset 2CBFS and Dataset 3IGFS, in terms of classification accuracy, precision, recall and f-measure.

Figure 4.3 shows the performance of the five ML classifiers based on the classification accuracy on each of the 6 features applied in this research. The classification accuracies of the five ML classifiers were increased with the complete set of 6 features Dataset 1 as shown in figure 4.4. Where the highest classification accuracy of 99.48% was achieved by the Random Forest. Moreover, SVM produced the lowest accuracy of 96.59%. MLP achieved a classification accuracy of 97.85%, while both KNN and decision tree achieved accuracies of over 98%.



	MLP	KNN	SVM	C4.5	Random forest
Total number of transmitted packets	88.59	90.74	71.85	93.7	90.96
Total number of received packets	85.92	86.14	66.66	90.59	86.22
Total Transmitted Kbytes	73.62	91.11	44.44	91.03	91.11
Total received Kbytes	68.74	84.74	44.66	91.33	85.11
Transmitting data rate in Kbytes/sec	63.11	76.29	63.25	77.25	76.14
Receiving data rate in Kbytes/sec	68.96	70.07	63.25	71.4	70.22

Figure 4. 3: Classification accuracy of ML classifiers on individual feature

Figure 4. 4: Classification accuracy of ML classifiers on dataset 1

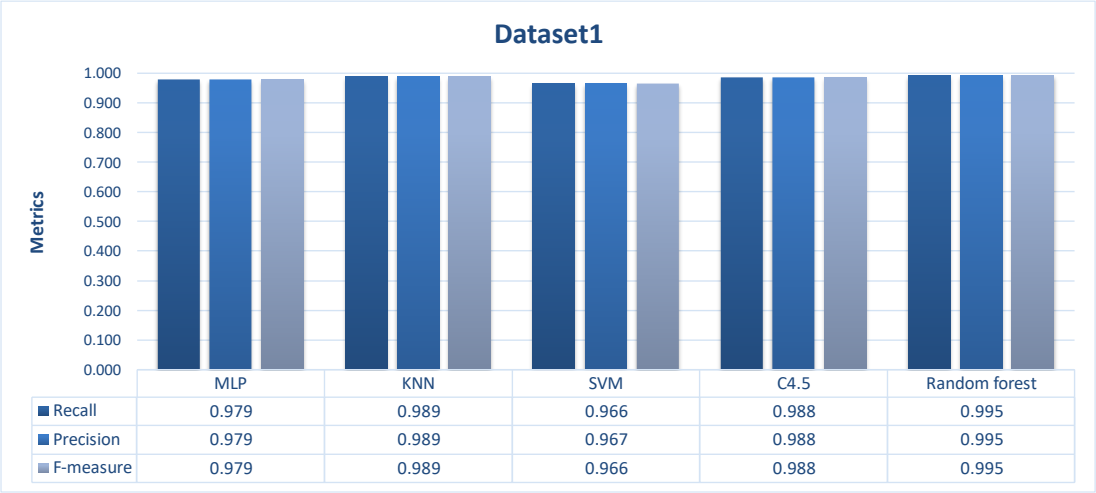


Figure 4. 5: Comparison of recall, precision and f-measure on dataset 1

In terms of recall, precision, and f-measure values of the 5 ML classifiers on Dataset 1, figure 4.5 shows that Random forest attained the highest values of 0.995 compared to all other classifiers. On the other hand, SVM had the lowest recall, precision, and f-measure values of 0.966, 0.967, and 0.966, respectively, when compared to all other classifiers.

Classification accuracy of the five ML classifiers on the reduced set of 4 features dataset 2CBFS; with the application of best first search method and consistency-based feature selection technique is represented in figure 4.6.

However, MLP produced the same classification accuracy of 97.85% as opposed to Dataset 1. Whereas the classification accuracy for the remaining ML classifiers was slightly decreased in Dataset 2CBFS compared to Dataset 1. It decreased by 0.14% for KNN, and 1.71% for SVM, while the accuracy of the decision tree decreased by 0.07% and 0.82% for the random forest.

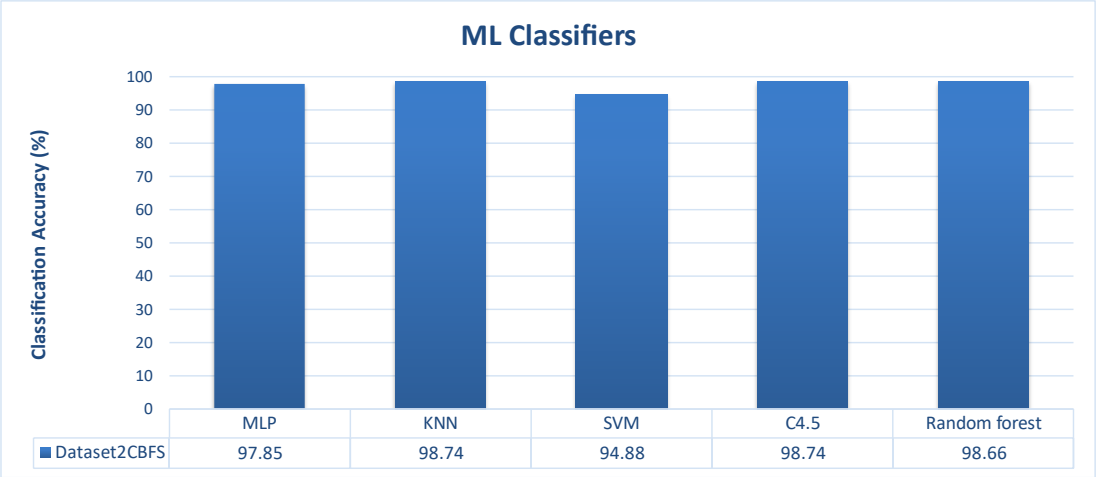


Figure 4. 6: Classification accuracy of ML classifiers on Dataset 2CBFS

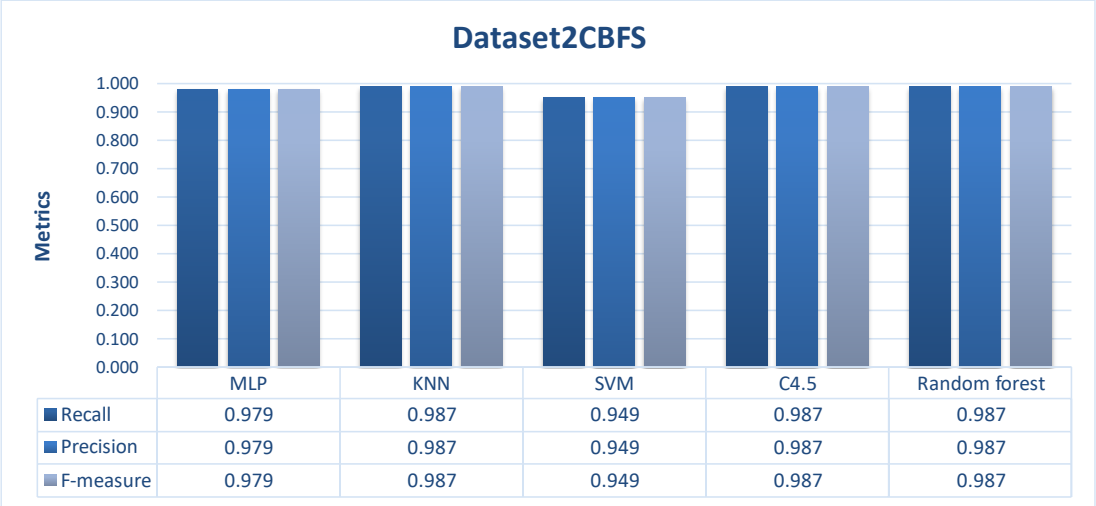


Figure 4. 7: Comparison of recall, precision and f-measure on dataset 2CBFS

Figure 4.7 shows the comparison of recall, precision, and f-measure values on dataset 2CBFS. The recall, precision, and f-measure values of 0.979 for MLP

remain unchanged as opposed to Dataset 1. While the metrics values of the rest of the ML classifiers slightly decreased in comparison to Dataset 1.

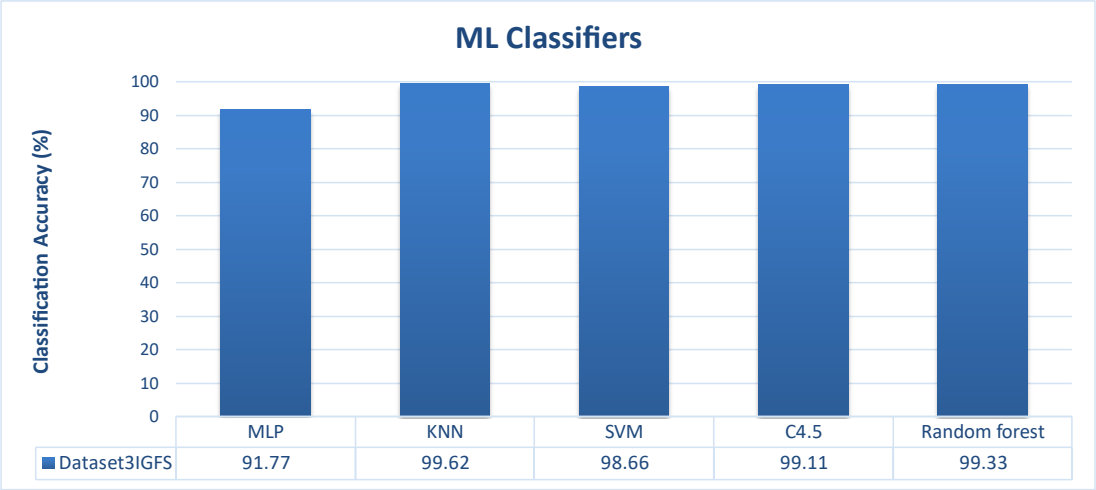


Figure 4. 8: Classification accuracy of ML classifiers on dataset 3IGFS

Figure 4.8 represents the Dataset 3IGFS, where the set of features extracted from the full set of 6 features using the information gain technique. The top 4 features were selected based on ranker method.

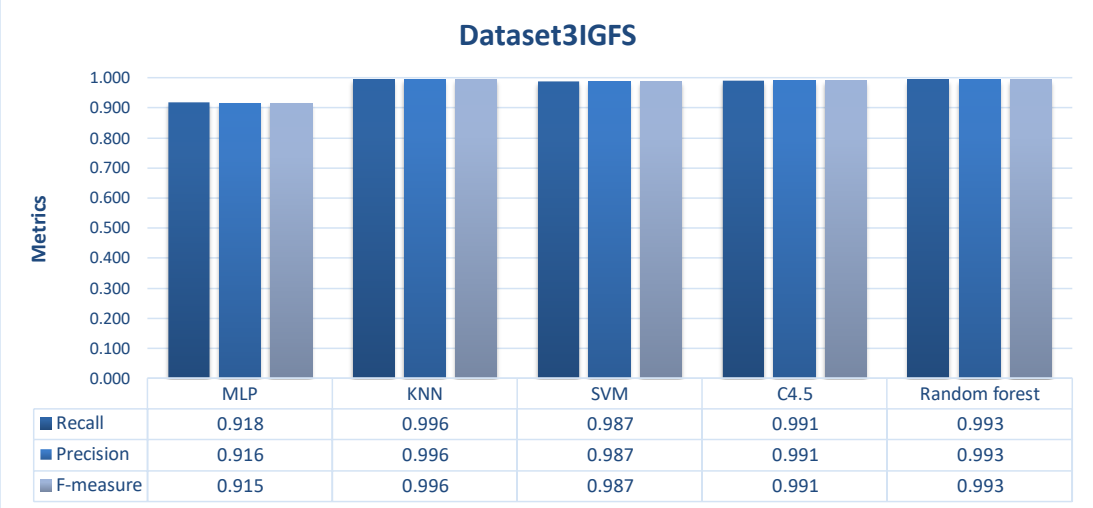
KNN achieved the highest classification accuracy of 99.62% in Dataset 3IGFS. Moreover, both SVM and decision tree produced better classification accuracy of 98.66% and 99.11% compared to dataset 1 and dataset 2CBFS.

Furthermore, the classification accuracy of 99.33% for Random forest was decreased compared to dataset 1. While the classification accuracy of 91.77% for MLP was the lowest in Dataset 3IGFS compared to Dataset 1 and Dataset 2CBFS.

Figure 4.9 displays the comparison of recall, precision and f-measure values of the five ML classifiers on Dataset 3IGFS.

The recall, precision, and f-measure value of 0.996 was the highest in Dataset 3IGFS and was achieved by KNN. Moreover, the metrics values of recall,

precision, and f-measure for MLP were the lowest in Dataset 3IGFS compared to Dataset 1 and Dataset 2CBFS. Whereas both SVM and decision tree produced better values in terms of recall precision and f-measure compared to dataset 1 and dataset 2CBFS.



	MLP	KNN	SVM	C4.5	Random forest
Recall	0.918	0.996	0.987	0.991	0.993
Precision	0.916	0.996	0.987	0.991	0.993
F-measure	0.915	0.996	0.987	0.991	0.993

Figure 4. 9: Comparison of recall, precision and f-measure on dataset 3IGFS

Comparing the results obtained for the five ML classifiers in all datasets in terms of all evaluation metrics, we found that a number of effective features can be considered to improve the overall results. Moreover, we conclude that the optimum results in terms of all evaluation metrics used in these experiments were achieved by KNN in Dataset 3IGFS using10-fold cross-validation. We determined KNN to be the most suitable ML classifier in terms of classifying smartphone applications' network traffic based on different levels of behaviour and interaction.

Narudin et al. [150] discussed, the time taken by classifiers to build a model is very crucial and affects the resource consumption of a wireless device. Thus, considering the processing time of classifiers to build a model is very important.

Table 4.6 shows the time taken by each of ML classifier in all datasets to build an ML classification model.

The processing time of 0.01s for KNN was the shortest time to build a model and remain identical in all datasets, while the processing time of 1.42 s for MLP to build a model in dataset 1 was the longest compared with all other classifiers in all datasets. Moreover, the model building time for MLP, SVM and Random forest decreased in Dataset 3IGFS when compared with Dataset 1 and Dataset 2CBFS. Finally, the processing time of 0.01 s for the decision tree remains identical in Dataset 2CBFS and Dataset 3IGFS. Overall, the computational cost of constructing a model across all datasets has a minimal effect, and for KNN, it remained identical and did not vary when using either four or six features.

Table 4. 6: Processing time to build the classification model (in seconds)

ML Classifier	Dataset 1	Dataset 2CBFS	Dataset 3IGFS
MLP	1.42	1.11	1.09
KNN	0.01	0.01	0.01
SVM	0.41	0.42	0.32
C4.5	0.02	0.01	0.01
Random forest	0.26	0.24	0.22

4.6 Summary

This chapter presented the CALI approach in which the WNIC with the aid of an ML classification model, sleeps and awakes based on the level of network activity of each application. It described how different levels of traffic behaviour and interactions of nine smartphone applications were contextually exploited for the classification by the application of ML classifiers. This chapter also has described how CALI power saving modes enable additional power saving opportunities by adjusting the WNIC to sleep and awake in accordance with the smartphone applications' network activity. This was followed by

describing the process of data extraction and preparation employed in this research for building the training data. The chapter also described the experimental settings used for traffic classification, including the description of parameter settings for the selected ML classification models. This was followed by the performance evaluation of five ML classifiers on Dataset 1, Dataset 2CBFS and Dataset 3IGFS using 10-fold cross-validation.

5

EXPERIMENTATION: ANALYSES AND DISCUSSIONS

5.1 Introduction

This chapter provides detailed experimentation, analyses and discussions to determine whether the selected classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data. For all the experiments conducted in this chapter, specifically in sections 5.2 to 5.5, the ML classifiers are trained using the default hyperparameter settings listed in table 4.5 as baseline models, and then in chapter six, sections 6.4 to 6.10, a hyperparameter optimisation process is conducted to further enhance the model's performance. Section 5.2 starts by illustrating the applications used for training and testing the classification models, where an app of each class was selected for training the selected ML classifiers and the generalisation capacity of the classification models was tested on different apps that were not included in training data. This section also provides an in-depth analysis of the network traffic for the selected applications used for training and testing, It then describes the experimental setup, followed by presenting the results and discussing the outcomes. Sections 5.3 to 5.5 follow the same structure described for 5.2, but with the following differences in training and testing data. Section 5.3 extends the training data by including a voice call application taken from the testing data. While section 5.4 reduces the training data and further assesses the generalisation capacity of learned classification models on

reduced training data. Section 5.5 further assesses the generalisation capacity by training ML classifiers on applications that were previously used for testing in sections 5.2 and 5.3 and assesses their generalisation performance on applications that were used for training in sections 5.2 and 5.3. Section 5.6 provides detailed conclusions based on previous sections. Furthermore, given that the classification models in sections 5.3 to 5.5 were capable of achieving high results on unseen testing data of applications that were not included in the training data. Thus, this section also explores the feasibility of manually crafting rules to hand-classify the training data. Where an attempt to hand-classify the training data is made, followed by a discussion and comparison of the outcomes with the classification models constructed using ML classifiers.

5.2 Training with an App of Each Class and Testing on Different App(s) of the Same Class

In this section, experiments are performed to determine whether the classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data.

Table 5.1 lists the selected applications that are used for training the ML classifiers, whereas Table 5.2 lists the applications that are used for testing the classification models.

Training set based on the following applications:

Table 5. 1: Training set 1

Applications	Class	Samples
1- Skype video call	High	150
2- Facebook	Varied	150
3- New Star Soccer (NSS)	Low	150
4- XiiaLive internet radio app	Buffer	150

Total Training Samples: 600

Testing set based on the following applications:

Table 5. 2: Testing set 1

Applications	Class	Samples
1- Google Hangouts video call		150
2- Google Hangouts Voice call	High	150
3- Skype voice call		150
4- Gmail	Varied	150
5- New Star Cricket (NSC)	Low	150

Total Testing Samples: 750

Moreover, figure 5.1 shows the receiving traffic of both, the listed applications in Table 5.1 that are used for training the ML classifiers and the listed applications in Table 5.2 that are used for testing the classification models. Whereas the transmitting traffic of previously mentioned applications is shown in figure 5.2.

By observing the receiving and transmitting traffic of both the training and testing data in figures 5.1 and 5.2, we can see the similarity between the Skype video call that is used for training the ML classifiers and the Google Hangouts video call that is used for testing the classification models.

This similarity is in terms of there being a clear separation between the training data of Skype video call and other applications in the training set, similarly in the testing data of Google Hangouts video call and other applications in the testing set. But, with a noticeable variation in the traffic range between the training data of Skype video call and the testing data of Google Hangouts video call.

So, by observing the receiving traffic in figure 5.1 it can be seen that the traffic range of Skype video call varies between 57.43 and 145 KBs, while a higher variation can be seen in the traffic range of Google Hangouts video call that varies between 50.17 and 521.25 KBs. This variation in the traffic range is also

similar to the transmitting traffic that can be seen in figure 5.2, where the traffic for Skype video call varies between 60.41 to 156.25 KBs and it varies between 32.31 and 471.25 KBs for Google Hangouts video call.

However, since there is a clear-cut separation between the traffic of video call applications and other applications in the training and testing data, therefore, it would be expected that the learned classification models on training data of Skype video call would be capable to capture the aforementioned variance in the traffic range and generalise to unseen testing data of Google Hangouts video call.

Furthermore, by observing the traffic of the voice call applications; Skype voice call and Google Hangouts voice call, an overlapping can be observed between the testing data of these applications and the testing data of applications of other classes. However, due to the differences in the traffic trends of the training data of Skype video call and the testing data of voice call applications, it is possible that the learned classification models on Skype video might be incapable to generalise to testing samples of both Skype and Google Hangouts voice calls.

Finally, by observing the training data in figures 5.1 and 5.2 an overlapping can be seen between the traffic of applications that belong to classes buffer, varied and low with more overlapping can be noticed between varied and low classes. Thus, it would be interesting to test the generalisation capacity of the learned classification models on overlapping training data, more specifically of classes varied and low.

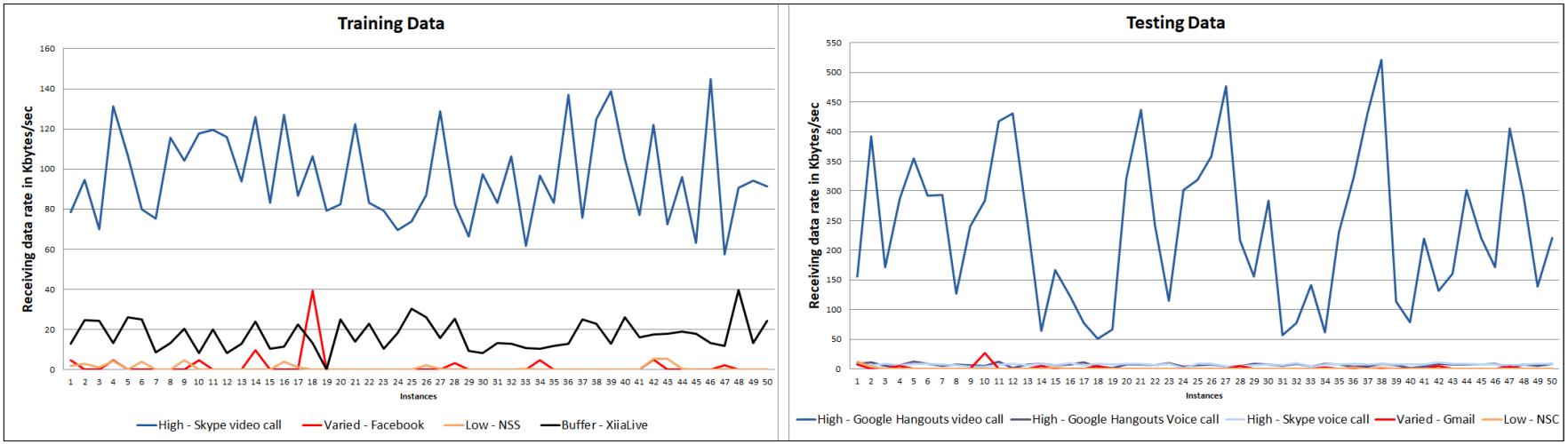


Figure 5. 1: Levels of network interaction of receiving traffic for apps listed in table 5.1 that are used for training and for apps listed in table 5.2 that are used for testing

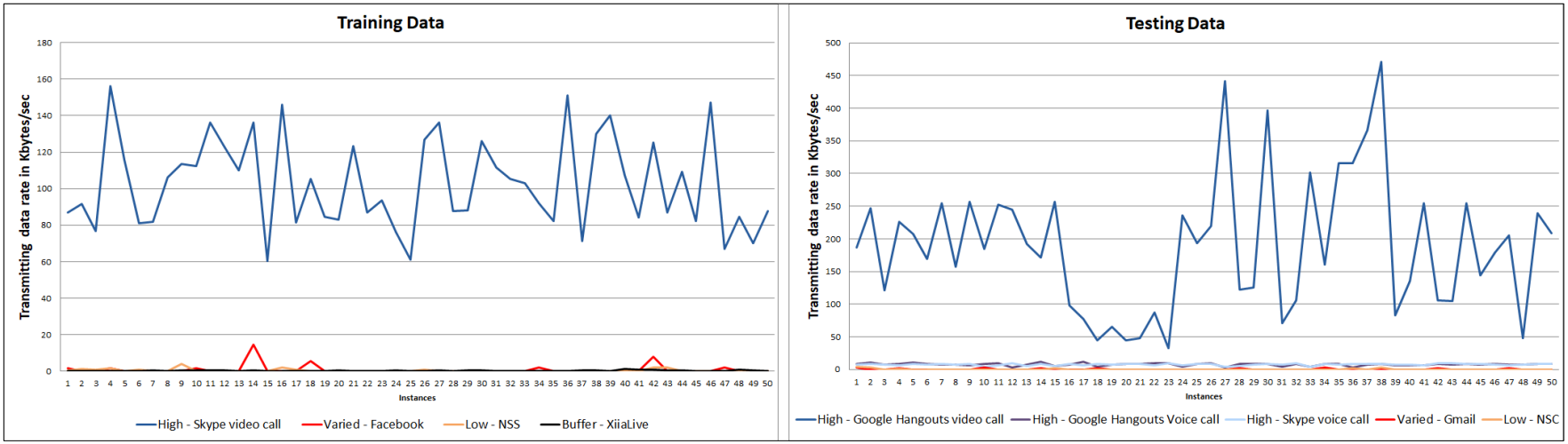


Figure 5. 2: Levels of network interaction of transmitting traffic for apps listed in table 5.1 that are used for training and for apps listed in table 5.2 that are used for testing

5.2.1 Experimental Setup

The following experiment was performed with the aid of WEKA tool. On a desktop computer operating Microsoft Windows 10, with Intel core i7-4800MQ CPU of 3.70 GHz and 12 GB of RAM.

To assess the generalisation capacity, the ML classifiers used in the 4.5 experiments were chosen. These are MLP, KNN, SVM, decision tree (C4.5), and Random Forest. The five ML classifiers were trained on samples with six features of applications listed in table 5.1, and their generalisation capacity was tested on samples of applications listed in table 5.2.

This experiment was carried out by training the chosen five ML classification models using WEKA's default parameter settings listed in table 4.5.

The performance of each classifier is evaluated in terms of classification accuracy, along with other evaluation metrics used for multiclass classification, such as confusion matrix, macro-average of precision, recall and weighted average f-measure.

The confusion matrix provides a detailed breakdown of the predictions, including the distribution of correct and incorrect predictions made by the classification models. Macro-average is the arithmetic mean of the per-class measures and is represented as a single value. Whereas weighted average f-measure takes into account the imbalanced distribution of classes, it calculates the F1-scores for each class and then weights them by the number of actual instances in each class [249, 251].

5.2.2 Results

5.2.2.1 Classification Model: MLP

=== Results ===

Correctly Classified Instances	445	59.33%
Incorrectly Classified Instances	305	40.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	149	159	141	1
	Varied	1	148	0	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.795.
Recall: 0.593.
Weighted Avg:
F-Measure: 0.562.

5.2.2.2 Classification Model: KNN

=== Results ===

Correctly Classified Instances	516	68.8%
Incorrectly Classified Instances	234	31.2%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	252	14	84	100
	Varied	1	148	0	1
	Low	0	0	116	34
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.896.
Recall: 0.688.
Weighted Avg:
F-Measure: 0.752.

5.2.2.3 Classification Model: SVM

=== Results ===

Correctly Classified Instances	536	71.46%
Incorrectly Classified Instances	214	28.53%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	246	202	0	2
	Varied	5	142	2	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.868.
Recall: 0.715.
Weighted Avg:
F-Measure: 0.733.

5.2.2.4 Classification Model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	371	49.46%
Incorrectly Classified Instances	379	50.53%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	144	0	201	105
	Varied	1	79	69	1
	Low	0	0	148	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.867.
Recall: 0.495.
Weighted Avg:
F-Measure: 0.533.

5.2.2.5 Classification Model: Random Forest

=== Results ===

Correctly Classified Instances	413	55.06%
Incorrectly Classified Instances	337	44.93%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	147	7	136	160
	Varied	0	149	0	1
	Low	0	0	117	33
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.884.
Recall: 0.551.
Weighted Avg:
F-Measure: 0.606.

5.2.3 Discussion

The previous experiment was performed to determine whether the learned classification models on training data of applications listed in table 5.1 are able to generalise to testing data of applications listed in table 5.2.

Firstly, in terms of correctly classifying class high samples. By observing the confusion matrices of all the classification models, it can be seen that testing samples of class high were mainly misclassified into other classes. This is because the learned classification models were only capable of capturing the

variance in the traffic range of video call applications and that by generalising well to unseen testing samples of Google Hangouts video call only.

Moreover, since Skype voice call and Google Hangouts voice call were only included in the testing data and since the traffic of these applications is overlapping with the traffic of applications of other classes. As a result, the resultant classification models were incapable to generalise to testing samples of both Skype and Google Hangouts voice calls, and thus classifying them mostly into other classes. Consequently, the training data should contain a wider range of sample data.

Furthermore, SVM has comparatively produced better classification results in terms of correctly classifying the testing samples from all classes with an overall accuracy of 71.46%. As outlined in subsection 3.6.6.5, this is due to the presence of a lower degree of non-linearity in the training and testing data overall, along with the strength of applying the normalised polynomial kernel which normalises the kernel values and improves the numerical stability, as a result, better performance was achieved by SVM compared to other classification models. Moreover, in terms of the macro average of precision, recall, and weighted f-measure. KNN achieved the highest macro average precision of 0.896 and weighted F-measure of 0.752. While SVM attained the highest macro average recall of 0.715.

Finally, in terms of assessing the generalisation capacity of the learned classification models on the overlapping training data of classes varied and low. By observing the confusion matrices, it can be seen that the classification models were able to capture the overlapping and that by achieving good generalisation performance on the testing data of these classes.

Since the resultant classification models were incapable to generalise to testing data of both Skype and Google Hangouts voice calls. Because none of the voice

call applications were included in the training data as both were only included in the testing data. Therefore, the following experiment will be performed using extended training data, which will include samples from the Skype voice call application.

5.3 Extending the Training Data by Including the Skype Voice Call Application

Table 5.3 lists the applications that are used for training the ML classifiers, whereas table 5.4 lists the applications that are used for testing the classification models.

Training set based on the following apps:

Table 5. 3: Training set 2

Applications	Class	Samples
1- Skype video call	High	150
2- Skype voice call		150
3- Facebook	Varied	150
4- New Star Soccer (NSS)	Low	150
5- XiiaLive internet radio app	Buffer	150

Total Training Samples: 750

Testing set based on the following apps:

Table 5. 4: Testing set 2

Applications	Class	Samples
1- Google Hangouts video call	High	150
2- Google Hangouts Voice call		150
3- Gmail	Varied	150
4- New Star Cricket (NSC)	Low	150

Total Testing Samples: 600

In addition, figure 5.3 shows the receiving traffic of both, the listed applications in table 5.3 that are used for training the ML classifiers and the listed applications in table 5.4 that are used for testing the classification models. Whereas the transmitting traffic of previously mentioned applications is shown in figure 5.4.

By observing the receiving traffic of the training data in figure 5.3, a higher variation can be seen in the range of traffic after the inclusion of Skype voice call into the training data. So, when the receiving traffic of both Skype video call and Skype voice call is combined it varies between 0.76 to 145 KBs, while this varies between 0.79 and 521.25 KB in the testing data when the receiving traffic of Google Hangouts video call and Google Hangouts voice call is combined.

This variation in the traffic range is also similar to the transmitting traffic that can be noticed in Figure 5.4, where the combined transmitting traffic of Skype video call and Skype voice call varies between 3.69 to 156.25 KBs, and in the testing data, it varies between 2.93 and 471.25 KBs when the transmitting traffic of Google Hangouts video call and Google Hangouts voice call is combined.

Moreover, by observing the receiving traffic of the training data, an overlapping can be seen between the training data of Skype voice call and the training data of applications of other classes. While this also can be noticed in the receiving traffic of the testing data of Google Hangouts voice call.

Furthermore, by observing the transmitting traffic of the training data in figure 5.4, slighter overlapping can be seen between the training data of Skype and class varied, while no overlapping can be seen between the testing data of Google Hangouts voice call and the testing data of applications of other classes.

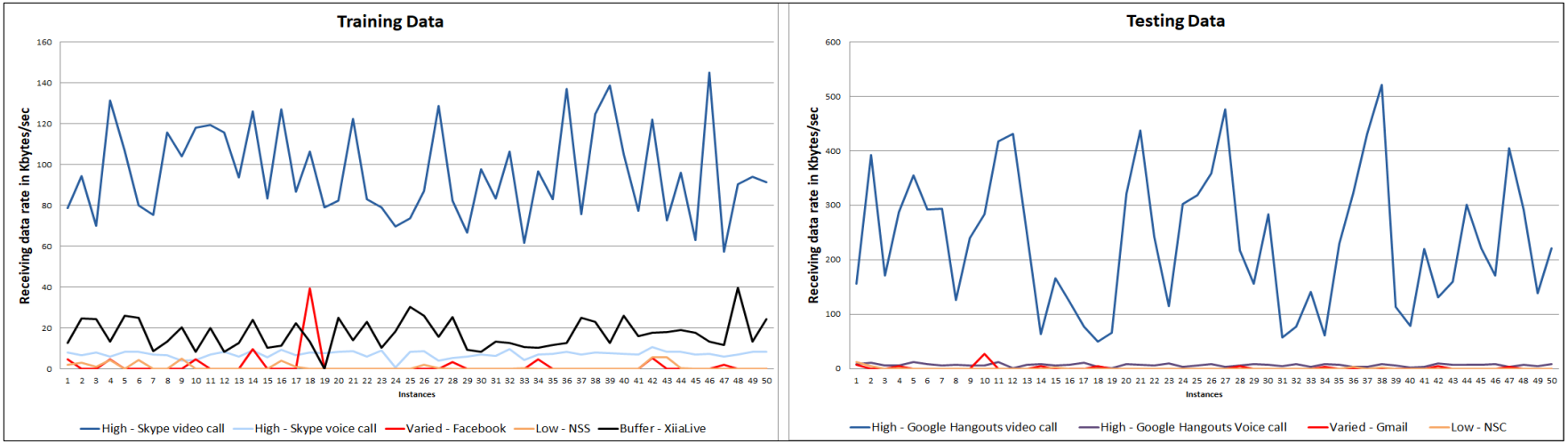


Figure 5. 3: Levels of network interaction of receiving traffic for apps listed in table 5.3 that are used for training and for apps listed in table 5.4 that are used for testing

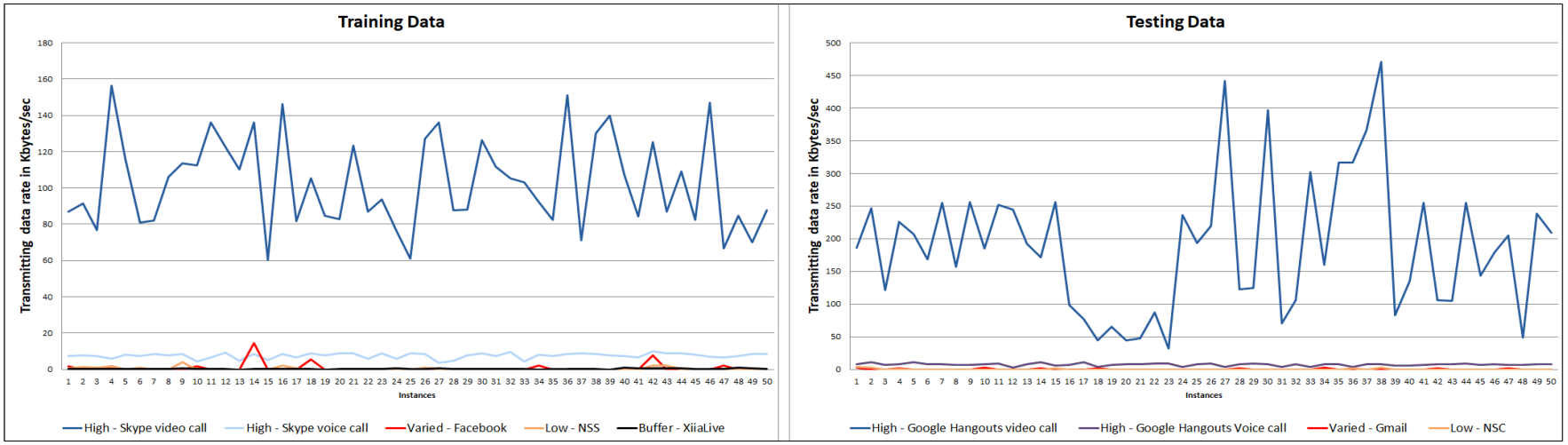


Figure 5. 4: Levels of network interaction of transmitting traffic for apps listed in table 5.3 that are used for training and for apps listed in table 5.4 that are used for testing

Finally, it would be interesting to test the generalisation capacity of the learned classification models on training data of applications listed in table 5.3, more specifically after the inclusion of the Skype voice call application into the training data.

5.3.1 Experimental Setup

The setup of this experiment remains the same as in 5.2.1, where the five selected ML classifiers MLP, KNN, SVM, decision tree (C4.5), and Random Forest are trained using the default hyperparameter settings listed in table 4.5. The only difference is in the training and testing data of the applications being used. Where tables 5.3 and 5.4 list the applications used for training and testing the classification models.

Moreover, similar to experiment 5.2.1, the performance of each classifier is evaluated in terms of classification accuracy, macro-average of precision, recall and weighted average f-measure. Additionally, a confusion matrix is provided to examine the distribution of correct and incorrect predictions made by the classifiers.

5.3.2 Results

5.3.2.1 Classification Model: MLP

=== Results ===

Correctly Classified Instances	572	95.33%
Incorrectly Classified Instances	28	4.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	279	4	0	17
	Varied	1	148	0	1
	Low	0	0	145	5
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.953.
Weighted Avg:
F-Measure: 0.972.

5.3.2.2 Classification Model: KNN

=== Results ===

Correctly Classified Instances	561	93.5%
Incorrectly Classified Instances	39	6.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	297	1	2	0
	Varied	1	148	0	1
	Low	0	0	116	34
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.935.
Weighted Avg:
F-Measure: 0.961.

5.3.2.3 Classification Model: SVM

=== Results ===

Correctly Classified Instances	562	93.66%
Incorrectly Classified Instances	38	6.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	272	26	0	2
	Varied	5	142	2	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.948.
Recall: 0.937.
Weighted Avg:
F-Measure: 0.941.

5.3.2.4 Classification Model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	494	82.33%
Incorrectly Classified Instances	106	17.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	298	0	0	2
	Varied	1	79	69	1
	Low	0	0	117	33
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.906.
Recall: 0.823.
Weighted Avg:
F-Measure: 0.844.

5.3.2.5 Classification Model: Random Forest

=== Results ===

Correctly Classified Instances	548	91.33%
Incorrectly Classified Instances	52	8.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	130	19	0
	Low	0	0	118	32
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.964.
Recall: 0.913.
Weighted Avg:
F-Measure: 0.937.

5.3.3 Discussion

The previous experiment was performed by training the ML classifiers on training data of applications listed in table 5.3, and the generalisation capacity of the classification models was tested on unseen testing data of applications listed in table 5.4.

So, based on the experimental results, it can be seen that the learned classification models on extended training data that included Skype voice call, were capable of achieving good generalisation performance more specifically on testing samples of class high with an overall accuracy ranging from 82.33% for C4.5 to 95.33% for MLP. Moreover, both MLP and KNN achieved the highest weighted average f-measure of 0.972 and 0.961 respectively, while the lowest weighted average f-measure of 0.844 was achieved by decision tree.

This improvement in the generalisation performance was due to the training of the ML classifiers on a wider variation range that was noticed in the receiving and transmitting traffic in figures 5.3 and 5.4 after the inclusion of Skype voice call into the training data, where the learned classification models were capable to capture this variance by generalising well to unseen testing data of class high.

However, since the classification models have consistently demonstrated strong generalisation capabilities, the following experiment will be conducted by training the models on reduced training data to determine if they can still achieve good generalisation performance on the testing data.

5.4 Reducing the Training Data by Half and then by a Quarter

The following experiment is performed to assess the generalisation capacity of the learned classification models on a reduced training data. This would be carried on by reducing the amount of the training samples of each application listed in table 5.3 to half and then to the quarter while keeping the size of the testing samples the same.

Thus, out of 750 training samples that were used for training the ML classifiers in the second experiment, the ML classifiers will be trained with 375 samples and then with 185 samples in this experiment. Finally, their generalisation performance will be assessed on applications that are listed in table 5.4.

Training set based on the following apps:

Table 5. 5: (Training set 3 of 185 samples), (Training set 4 of 375 samples)

Applications	Class	Samples
1- Skype video call	High	75, 37
2- Skype voice call		75, 37
3- Facebook	Varied	75, 37
4- New Star Soccer (NSS)	Low	75, 37
5- XiiaLive internet radio app	Buffer	75, 37

Total Training Samples: 375, 185

Testing set based on the following apps:

Table 5. 4: Testing set 2

Applications	Class	Samples
1- Google Hangouts video call	High	150
2- Google Hangouts Voice call		150
3- Gmail	Varied	150
4- New Star Cricket (NSC)	Low	150

Total Testing Samples: 600

5.4.1 Experimental Setup

The setup of this experiment remains the same as in 5.2.1, where the five selected ML classifiers MLP, KNN, SVM, decision tree (C4.5), and Random Forest are trained using the default hyperparameter settings listed in table 4.5.

Moreover, the training and testing data of applications remain the same as in 5.3.1, with differences in the amount of training data being used for training. Table 5.5 lists the applications and the amount of training data used for training. While table 5.4 lists the applications that are used for testing the classification models.

Furthermore, similar to experiment 5.2.1, the performance of each classifier is evaluated in terms of classification accuracy, macro-average of precision, recall and weighted average f-measure. Additionally, a confusion matrix is provided to examine the distribution of correct and incorrect predictions made by the classifiers.

5.4.2 Results

5.4.2.1 Classification Model: MLP

Training samples: 750

=== Results ===

Correctly Classified Instances	572	95.33%
Incorrectly Classified Instances	28	4.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	279	4	0	17
	Varied	1	148	0	1
	Low	0	0	145	5
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.953.
Weighted Avg:
F-Measure: 0.972.

Training samples: 375

=== Results ===

Correctly Classified Instances	530	88.33%
Incorrectly Classified Instances	70	11.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	237	0	0	63
	Varied	1	148	0	1
	Low	0	0	145	5
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.998.
Recall: 0.883.
Weighted Avg:
F-Measure: 0.935.

Training samples: 185

=== Results ===

Correctly Classified Instances	486	81%
Incorrectly Classified Instances	114	19%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	193	0	46	61
	Varied	1	147	0	2
	Low	0	0	146	4
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.938.
Recall: 0.810.
Weighted Avg:
F-Measure: 0.810.

5.4.2.2 Classification Model: KNN

Training samples: 750

=== Results ===

Correctly Classified Instances	561	93.5%
Incorrectly Classified Instances	39	6.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	297	1	2	0
	Varied	1	148	0	1
	Low	0	0	116	34
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.935.
Weighted Avg:
F-Measure: 0.961.

Training samples: 375

=== Results ===

Correctly Classified Instances	564	94%
Incorrectly Classified Instances	36	6%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	148	0	1
	Low	0	0	116	34
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.998.
Recall: 0.940.
Weighted Avg:
F-Measure: 0.966.

Training samples: 185

=== Results ===

Correctly Classified Instances	592	98.66%
Incorrectly Classified Instances	8	1.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	148	0	1
	Low	0	0	144	5
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.997.
Recall: 0.987.
Weighted Avg:
F-Measure: 0.992.

5.4.2.3 Classification Model: SVM

Training samples: 750

=== Results ===

Correctly Classified Instances	562	93.66%
Incorrectly Classified Instances	38	6.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	272	26	0	2
	Varied	5	142	2	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.948.
Recall: 0.937.
Weighted Avg:
F-Measure: 0.941.

Training samples: 375

=== Results ===

Correctly Classified Instances	550	91.66%
Incorrectly Classified Instances	50	8.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	264	32	0	4
	Varied	5	140	2	3
	Low	0	0	146	4
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.941.
Recall: 0.917.
Weighted Avg:
F-Measure: 0.926.

Training samples: 185

=== Results ===

Correctly Classified Instances	482	80.33%
Incorrectly Classified Instances	118	19.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	210	87	0	3
	Varied	11	125	2	12
	Low	0	0	147	3
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.869.
Recall: 0.803.
Weighted Avg:
F-Measure: 0.822.

5.4.2.4 Classification Model: Decision tree (C4.5)

Training samples: 750

=== Results ===

Correctly Classified Instances	494	82.33%
Incorrectly Classified Instances	106	17.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	298	0	0	2
	Varied	1	79	69	1
	Low	0	0	117	33
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.906.
Recall: 0.823.
Weighted Avg:
F-Measure: 0.844.

Training samples: 375

=== Results ===

Correctly Classified Instances	425	70.83%
Incorrectly Classified Instances	175	29.16%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	0	1	0
	Varied	0	41	107	2
	Low	64	0	85	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.772.
Recall: 0.708.
Weighted Avg:
F-Measure: 0.682.

Training samples: 185

=== Results ===

Correctly Classified Instances	465	77.5%
Incorrectly Classified Instances	135	22.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	291	1	8	0
	Varied	1	27	20	102
	Low	2	0	147	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.946.
Recall: 0.775.
Weighted Avg:
F-Measure: 0.792.

5.4.2.5 Classification Model: Random Forest

Training samples: 750

=== Results ===

Correctly Classified Instances	548	91.33%
Incorrectly Classified Instances	52	8.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	130	19	0
	Low	0	0	118	32
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.964.
Recall: 0.913.
Weighted Avg:
F-Measure: 0.937.

Training samples: 375

=== Results ===

Correctly Classified Instances	488	81.33%
Incorrectly Classified Instances	112	18.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	2	72	26	50
	Low	33	0	116	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.902.
Recall: 0.813.
Weighted Avg:
F-Measure: 0.833.

Training samples: 185

=== Results ===

Correctly Classified Instances	470	78.33%
Incorrectly Classified Instances	130	21.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	27	5	117
	Low	6	0	143	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.980.
Recall: 0.783.
Weighted Avg:
F-Measure: 0.810.

5.4.3 Discussion

The previous experiment was performed to assess the generalisation capacity of the learned classification models on reduced training data, where the training samples of each application listed in table 5.5 were reduced to half and then to a quarter.

Based on the experimental results, it can be observed that, reducing the amount of training data has a minimal impact on the generalisation performance of classification models, as they still were capable of achieving good generalisation performance on the testing data.

Moreover, in contrast to other classification models, an increase in the generalisation performance of the KNN was observed when the training samples kept decreasing. As outlined in subsections 3.5.1 and 3.6.7, in the presence of a small number of features and limited training data, KNN can be more efficient. This is because, unlike other classifiers, KNN has no explicit learning phase. Instead during the testing phase, it searches through the training data for the most similar or nearest instances in the region of k-neighbourhood, and since there is an overall similarity between the training and testing data, in this case, the nearest neighbours to a test instance are similar and more representative of the overall distribution of the data, as a result, better generalisation performance was achieved by KNN when the training samples were reduced. In addition, the KNN classification model achieved the highest macro average of precision 0.998, recall 0.940 and weighed average of f-measure 0.966, when it was trained on a reduced training data of a quarter of the size.

Furthermore, a slight increase in the generalisation performance of the decision tree was observed when the amount of training data was reduced from 375 samples to 185 samples. This matches what was outlined in

subsection 3.6.7 in relation to unstable classifiers, such as decision trees, that are sensitive to training data manipulation, where a small modification in training data leads to a significant change in the constructed classifier and therefore results in large differences in the prediction results. So, with 185 samples different rules were generated, whereby the algorithm generated fewer but more generalisable rules to the testing data, whereas the generated rules with 375 samples were more specific to the training data but less generalisable to the testing data.

5.5 Further Assessment of the Generalisation Capacity

To further assess the generalisation capacity, the following experiment will be conducted by switching the training and testing data of applications that were used in the previous experiments 2 and 3. This involves training the ML classifiers on applications that were previously used for testing and assessing their generalisation performance on applications that were used for training.

Training set based on the following apps:

Table 5. 6: Training set 5

Applications	Class	Samples
1- Google Hangouts video call	High	150
2- Google Hangouts Voice call		150
3- Gmail	Varied	150
4- New Star Cricket (NSC)	Low	150
5- XiiaLive internet radio app	Buffer	150

Total Training Samples: 750

Testing set based on the following apps:

Table 5. 7: Testing set 3

Applications	Class	Samples
1- Skype video call	High	150
2- Skype voice call		150
3- Facebook	Varied	150
4- New Star Soccer (NSS)	Low	150

Total Testing Samples: 600

5.5.1 Experimental Setup

The setup of this experiment remains the same as in 5.2.1, where the five selected ML classifiers MLP, KNN, SVM, decision tree (C4.5), and Random Forest are trained using the default hyperparameter settings listed in table 4.5. The only difference is in the training and testing data of the applications being used. Where tables 5.6 and 5.7 list the applications used for training and testing the classification models. Moreover, similar to experiment 5.2.1, the performance of each classifier is evaluated in terms of classification accuracy, macro-average of precision, recall and weighted average f-measure. Additionally, a confusion matrix is provided to examine the distribution of correct and incorrect predictions made by the classifiers.

5.5.2 Results

5.5.2.1 Classification Model: MLP

=== Results ===

Correctly Classified Instances	586	97.66%
Incorrectly Classified Instances	14	2.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	292	8	0	0
	Varied	0	148	0	2
	Low	0	2	146	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.984.
Recall: 0.977.
Weighted Avg:
F-Measure: 0.980.

5.5.2.2 Classification Model: KNN

=== Results ===

Correctly Classified Instances	589	98.16%
Incorrectly Classified Instances	11	1.83%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	296	0	0	4
	Varied	3	147	0	0
	Low	0	2	146	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.982.
Weighted Avg:
F-Measure: 0.987.

5.5.2.3 Classification Model: SVM

=== Results ===

Correctly Classified Instances	558	93%
Incorrectly Classified Instances	42	7%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	24	121	0	5
	Low	1	2	137	10
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.957.
Recall: 0.930.
Weighted Avg:
F-Measure: 0.940.

5.5.2.4 Classification Model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	597	99.5%
Incorrectly Classified Instances	3	0.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	1	0	0
	Varied	0	150	0	0
	Low	1	1	148	0
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.995.
Recall: 0.995.
Weighted Avg:
F-Measure: 0.995.

5.5.2.5 Classification Model: Random Forest

=== Results ===

Correctly Classified Instances	595	99.16%
Incorrectly Classified Instances	5	0.83%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	2	148	0	0
	Low	1	2	147	0
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.992.
Recall: 0.992.
Weighted Avg:
F-Measure: 0.992.

5.5.3 Discussion

To further assess the generalisation capacity, this experiment was performed by switching the training and testing data that were used in the previous experiments 2 and 3. This involved training the ML classifiers on applications that were previously used for testing and assessing their generalisation performance on applications that were used for training.

The obtained results showed an improvement in the generalisation performance of all classification models, this is due to the training of the classifiers on a wider variation range which therefore improved the generalisation performance on unseen data.

To illustrate this further, by referring to the receiving traffic of testing data in figure 5.3 a wider variation in the traffic range can be seen when the testing data of Google Hangouts video and Google Hangouts voice are combined, compared to the combined training data of Skype video call and Skype voice call. This variability is also apparent in the transmitting traffic of these applications, as shown in Figure 5.4. Thus, training the ML classifiers on training data with a wider variation in the traffic range leads to better

generalisation performance. Moreover, the decision tree achieved a macro average of precision, recall, and weighted average f-measure of 0.995.

5.6 Conclusions

The previous experiments were performed to determine whether the classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data.

In terms of generalising to unseen testing data of class high, the results of the first experiment showed that the learned classification models were only capable of capturing the variance in the traffic range of video call applications and that by generalising well to unseen testing data of Google Hangouts video call only. But, in terms of the voice call applications, the resultant classification models were incapable to generalise to testing data of both Skype and Google Hangouts voice calls, this is because none of the voice call applications were included in the training data as both were only included in the testing data and the traffic of these applications was overlapping with the traffic of applications of other classes.

However, SVM has comparatively produced better classification results in terms of correctly classifying the testing samples from all classes. And that matches what was outlined in subsection 3.6.6.5 about the strength of applying the kernel and with the application of the normalised polynomial kernel, the kernel values are normalised which improves the numerical stability, as a result, better performance was achieved by SVM compared to other classification models.

Moreover, in terms of generalising to unseen testing data of classes varied and low, the obtained results showed that the classification models were able to

capture the overlapping and that by achieving good generalisation performance on the testing data of these classes.

The second experiment was performed to assess the generalisation capacity of the learned classification models on extended training data, particularly after the inclusion of the Skype voice call application into the training data. Where the experimental results showed an improvement in the generalisation performance specifically on testing data of class high, this was due to the training of the ML classifiers on a wider variation range that resulted from combining the training data of Skype video call and Skype voice call.

The third experiment was performed to assess the generalisation capacity of the learned classification models on reduced training data. Where the training samples of each application used for training the ML classifiers in the second experiment were reduced to half and then to a quarter. However, the results of this experiment showed that reducing the amount of training data has a minimal impact on the generalisation performance, but still, better generalisation performance can be achieved by training with more samples.

However, in contrast to other classification models, an increase in the generalisation performance of the KNN was observed when the training samples kept decreasing. This matches what was outlined in subsections 3.5.1 and 3.6.7, in the presence of a small number of features and limited training data, KNN can be more efficient. This is because, unlike other classifiers, KNN has no explicit learning phase. Instead during the testing phase, it searches through the training data for the most similar or nearest instances in the region of k-neighbourhood, and since there is an overall similarity between the training and testing data, in this case, the nearest neighbours to a test instance are similar and more representative of the overall distribution of the data, as a result, better generalisation performance was achieved by KNN when the training samples were reduced. Also, a slight increase in the generalisation

performance of the decision tree was observed when the amount of training data was reduced from 375 samples to 185 samples. This matches what was outlined in subsection 3.6.7 in relation to unstable classifiers, such as decision trees, that are sensitive to training data manipulation, where a small modification in training data leads to a significant change in the constructed classifier and therefore results in large differences in the prediction results. So, with 185 samples different rules were generated, whereby the algorithm generated fewer but more generalisable rules to the testing data, whereas the generated rules with 375 samples were more specific to the training data but less generalisable to the testing data.

Furthermore, the fourth experiment was performed to further assess the generalisation capacity. In this experiment, the ML classifiers were trained on applications that were previously used for testing, and their generalisation performance was tested on applications that were used for training. However, the results of this experiment showed that training the ML classifiers on training data with a wider variation in the traffic range leads to better generalisation performance.

Overall, the experimental results indicate that the classification models constructed using ML classifiers were capable of achieving good generalisation performance on testing data by recognising the overlapping and capturing all the patterns or trends from the training data, including the dominant one.

However, since the experiments conducted in sections 5.3 to 5.5, specifically in section 5.4, where the learned classification models on reduced training data were still capable of achieving high results on unseen testing data of applications that were not included in the training data. Therefore, the following attempt would be to hand-classify the training data by crafting the rules manually.

So, by observing the receiving and transmitting traffic of training data in figures 5.3 and 5.4, the following rules can be hand-crafted:

Rule 1: IF the application's receiving data rates = > 6 KBs AND transmitting data rates = > 6 KBs then the class = High.

Rule 2: IF the application's receiving data rates = > 10 KBs AND transmitting data rates <= 1 KBs then the class = Buffer.

Although there is an overlapping between samples of the Skype voice call application that belong to class high with samples belonging to other classes, however, it is feasible to classify applications belonging to classes high and buffer using hand-crafted rules which also can be further refined.

Moreover, by observing the training data in figures 5.3 and 5.4 it can be seen that samples belonging to classes; varied and low are extremely overlapping, thus it will be difficult to distinguish between samples of these classes by constructing a set of rules manually.

However, if we do construct the rules manually, then the results would be extremely overfitted to training samples. And here comes the role of ML, whereby algorithms learn from training data and produce a set of rules in an automated fashion. Therefore, the complexity of manually constructing a set of rules, more specifically for samples of classes varied and low, is handed to ML classifiers.

So, while it is feasible to hand-classify the training data by crafting the rules manually, however, classification models constructed using ML classifiers introduce the following:

Automatic rule generation: Faster development

Defining a set of rules manually for the entire training data can be challenging and time-consuming, as noted by observing the training data in figures 5.3 and 5.4, regardless of overlapping between the classes, however, it was feasible to define rules for training data belonging to classes high and buffer in an attempt to hand-classify training data of these classes.

Moreover, by observing the training data belonging to classes varied and low in aforementioned figures 5.3 and 5.4, it can be seen that the training data are highly overlapping, so it was difficult to distinguish between the training data of these two classes and define a set of rules that hand-classify the training data. Since there is no clear-cut boundary between the classes that can be translated into a set of predefined rules.

This is in contrast to the ML approach, where the complexity of manually constructing a set of rules is handed over to the ML, whereby an ML classifier such as a decision tree does not require manual labour to create and maintain a set of rules, instead, it automatically learns to create its own set of if-then statements based on patterns it finds in training data, which significantly accelerates and simplifies the development process with less human intervention.

Generalisation

In terms of generalisation, the ML algorithms are designed with the ability to learn different variations and relationships in the data. Thus, the ability to learn the underlying patterns and relationships enables ML-based algorithms to generalise on previously unseen data in an effective manner.

In contrast, a predefined set of rules can be prone to overfitting, more specifically in case there are no clear-cut decision boundaries between the

classes. In such cases, defining a set of rules can lead to overfitting, where these rules might perform well on training data, but not on new and unseen data since no consideration of the complex relationships and patterns in the data have been taken into account.

5.7 Summary

This chapter provided detailed experimentation, analyses and discussions to determine whether the selected classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data. This chapter also provided an in-depth analysis of the network traffic for the selected applications used in training and testing. To assess the generalisation capacity of the selected classification model, four main experiments were conducted in this chapter. For each experiment, an experimental setup, results, and discussion were provided. In the first experiment, a representative application from each class was selected for training the ML classifiers, and their generalisation capacity was evaluated on different applications that were not included in the training data. The second experiment was performed to assess the generalisation capacity of the learned classification models on extended training data, particularly after the inclusion of the Skype voice call application into the training data. While the third experiment was performed to assess the generalisation capacity of the learned classification models on reduced training data. Where the training samples of each application used for training the ML classifiers in the second experiment were reduced to half and then to a quarter. To further assess the generalisation capacity, the fourth experiment was performed by switching the training and testing data that were used in the previous experiments 2 and 3. This involved training the ML classifiers on applications that were previously used for testing and assessing their generalisation performance on applications that were used for training. This chapter also provided detailed conclusions based on conducted experiments, since the classification models

achieved high results on unseen testing data of applications not included in the training data, this chapter further explored the feasibility of manually crafting rules to hand-classify the training data. Where an attempt was made to hand-classify the training data, followed by a discussion and comparison of the outcomes with the classification models constructed using ML classifiers.

6

HYPERPARAMETER OPTIMISATION

6.1 Introduction

This chapter conducts a hyperparameter optimisation process to identify the optimal settings that result in a better-performing classification model. Section 6.2 overviews the common hyperparameter tuning methods. Section 6.3 describes the experimental setup carried out to perform the hyperparameter optimisation process. This section also justifies the selection of the chosen hyperparameter tuning methods employed in this chapter. This is Followed by conducting the hyperparameter optimisation process using both manual and automated tuning methods in sections 6.4 to 6.8. This is carried out by performing 10-fold cross-validation on the training data of the applications listed in table 5.5 of experiment three section 5.4 consisting of 185 samples. And then evaluating the performance of the constructed classification models using the obtained optimal sets of hyperparameter values on the testing data of the applications listed in table 5.4. Section 6.9 repeats the previous four experiments conducted in chapter five, sections 5.2 to 5.5 using the optimal sets of hyperparameter values that were obtained in sections 6.4 to 6.8 for the five classification models. However, the experimental results in section 6.9, particularly of the repeated experiments one and four indicated that using the optimised hyperparameters for a particular training data may not always lead to improved model performance when there are changes in the overall distribution of new training data. Therefore, in section 6.10 the optimal sets of hyperparameter values for classification models for the first and fourth experiments of (sections 5.2 and 5.5) are determined. The same method for

identifying the optimal sets of hyperparameter values described in 6.3 and employed in sections 6.4 to 6.8 is applied to determine the optimal settings that result in better-performing classification models. Where 10-fold cross-validation is performed to explore different hyperparameter settings on the training data of the applications listed in table 5.1 of the first experiment and table 5.6 of the fourth experiment. This is followed by evaluating the performance of the constructed classification models on the testing data of the applications listed in table 5.2 of the first experiment and table 5.7 of the fourth experiment. Section 6.11 provides a deeper analysis of the confusion matrix, and then describes the process of reweighting the training inputs by assigning costs to the class misclassifications in the cost matrix.

6.2 Hyperparameter Settings

The performance of ML models is highly dependent on the selection of the most appropriate hyperparameter values, where tuning these hyperparameters and finding the most appropriate setting can and often leads to a better-performing model [112, 113]. Some studies have shown that there is no single hyperparameter tuning method that can be deemed as the best [114], and some hyperparameter optimisation methods can yield accuracies similar to those obtained with default configurations [230, 231, 232]. While recent studies have shown that the Bayesian optimisation method can be more reliable for optimising the hyperparameter values, particularly in large search spaces [113, 233, 234, 235]. Moreover, ML practitioners tend to prefer manual tuning over other hyperparameter tuning methods since it increases their comprehension of the model [113, 236, 237]. Furthermore, when the computational resources are limited, research has also suggested using the default hyperparameter setting suggested by ML tools [230, 238].

However, it is a common practice to initially train the ML model using the default hyperparameter setting as the baseline model and subsequently

conduct a hyperparameter optimisation process to enhance the model's performance [239]. This was similarly followed in [240, 241, 242] where the authors of these studies initially trained the ML model using the default hyperparameter setting suggested by the WEKA tool and then conducted a hyperparameter optimisation process.

There are many methods that exist for optimising the hyperparameters, including manual tunning, grid search, random search, and Bayesian optimisation.

In manual tuning, a user tries different combinations of hyperparameter configurations based on personal knowledge or from the literature. Whereas the grid search method, searches for the optimal combination of hyperparameters by tying every parameter setting over a user-predefined range of hyperparameter values. While in a random search, the optimal combination of hyperparameters is searched by randomly sampling hyperparameter configurations from a user-predefined search space. Both grid search and random search treat each hyperparameter configuration independently. While in contrast, Bayesian optimisation determines the next set of hyperparameters to try by taking into account the previous results of tested hyperparameter values [112, 113, 239].

6.3 Experimental Setup

In the following sections 6.4 to 6.8, the hyperparameter optimisation process is performed to identify the optimal settings that result in a better-performing classification model. To explore different hyperparameter settings, 10-fold cross-validation was performed on the training data of experiment three section 5.4 consisting of 185 samples. This is followed by evaluating the performance of the constructed classification model on the testing data of the applications listed in table 5.4.

To identify the optimal hyperparameter values, we have utilised the manual tuning option, since it increases the comprehension of the model and allows us to understand the hyperparameter tuning effect in training and testing the classification models. The manual hyperparameter tuning is performed to optimise the hyperparameter values of MLP and SVM classification models.

Additionally, in a similar manner to the authors of [253, 254] who utilised both manual and automated tuning methods, WEKA's built-in parameter selection function called CVParameterSelection was utilised to automate the process of searching for optimal parameters. CVParameterSelection is a widely used method for tuning the hyperparameters adopted by many studies including [240, 243, 244, 245, 246]. This method involves searching through a user-specified range of values for the given parameters and identifying the optimal parameter values within that range. We used the CVParameterSelection to identify the optimal parameters of KNN, decision tree and Random forest using internal 10-fold cross-validation.

Finally, for the experiments in this chapter, except for the specified parameters that are used to identify the optimal values, all other parameter values for the five classification models remained unchanged as listed in table 4.5.

6.4 MLP Settings

6.4.1 Default Setting

To identify the optimal hyperparameter values of an MLP, the following most common hyperparameters are considered based on [247]: hidden layer (-H), learning rate (-L) and momentum (-M). While the range of hyperparameter values specified in [255] for L and M were considered. Moreover, since there are no clear rules to determine the optimal number of hidden layers and the number of nodes in each layer [248]. The tuning of the hidden layer parameter

(-H) is carried out using WEKA's four predefined wildcards of the hidden layer described in the following subsection.

In WEKA the parameter (-H) represents a number of hidden layers and the number of nodes in each layer, where the default setting of this parameter in WEKA is “a” which creates a network with a single hidden layer and the number of nodes = (number of features + number of classes) /2.

Thus, in our case with 6 features the default setting for the number of nodes would be 6 + 4 / 2 = 5 nodes in a single hidden layer.

The Learning rate hyperparameter of the backpropagation algorithm (-L) determines the size of the steps the weights are updated during training. The value of this hyperparameter ranges between 0 to 1, a higher learning rate values allow the model to learn faster but at a risk of overshooting the optimal weights. In contrast, lower learning rate values require longer training but are more likely to converge to the optimal weights. The default value for L in WEKA is 0.3.

During the training, the momentum parameter (-M) is added to speed up the optimisation algorithm's convergence towards the global minimum. Since the learning rate determines the size of the steps that are taken towards the minimum of the loss function during the gradient descent. During the weight updates, the value of the momentum parameter between 0 to 1 is added to determine how much influence the previous weight update has on the current weight update, in case the M value set to 0.9 means the current weight update is strongly influenced by the previous weight update. However, this also enables the algorithm to maintain a more consistent direction for these updates and thus accelerates the convergence towards the global minimum of the loss function. The default value of M in WEKA = 0.3.

Table 6.1 shows classification results obtained using the default parameter settings of an MLP in WEKA listed in table 4.5.

Table 6. 1: MLP default setting

Results of training and validation using 10-fold cross validation	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Correctly classified Instances 179 - 96.75%. Incorrectly classified instances 6 - 3.24%.	Correctly classified instances 486 - 81%. Incorrectly classified instances 114 - 19%.

In the following subsections the optimal hyperparameter values for H, L and M will be manually determined. While the rest of the parameters remained unchanged as listed in table 4.5.

6.4.2 Hidden Layers

In this subsection, the tuning of the hidden layer parameter H is carried out using WEKA's four predefined wildcards of the hidden layer, which are as follows:

1. WEKA's default setting "a" as described in 6.4.1.

2. "o" which creates a network of a single hidden layer and the number of nodes equal to a number of classes, thus the number of nodes in our case would be 4 nodes in a single hidden layer.

3. "i" creates a network of a single hidden layer and the number of nodes equal to a number of features, thus the number of nodes in our case would be 6 nodes in a single hidden layer.

4."t" creates a network of a single hidden layer and the number of nodes = a number of features + a number of classes. Thus, in our case with 6 features, the number of nodes would be 6 + 4 = 10 nodes in a single hidden layer.

Moreover, the number of hidden layers and the nodes per layer can also be specified in WEKA, for example with the following values of 5,3,2. The number of hidden layers created for the network would be 3, with 5 nodes in layer 1, 3 nodes in layer 2 and 2 nodes in layer 3.

Table 6. 2: MLP hidden layers setting

Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Default		
"a" = 5 nodes in a single hidden layer.	Correctly classified instances 179 - 96.75%. Incorrectly classified instances 6 - 3.24%.	Correctly classified instances 486 - 81%. Incorrectly classified instances 114 - 19%.
"o" = 4 nodes in a single hidden layer.	Correctly classified instances 179 - 96.75% Incorrectly classified instances 6 - 3.24%	Correctly classified instances 460 – 76.66% Incorrectly classified instances 140 – 23.33%
"i" = 6 nodes in a single hidden layer.	Correctly classified instances 179 - 96.75% Incorrectly classified instances 6 - 3.24%	Correctly classified instances 466 – 77.66% Incorrectly classified instances 134 – 23.33%
"t" = 10 nodes in a single hidden layer.	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 495 – 82.5% Incorrectly classified instances 105 – 17.5%

6.4.2.1 Discussion

Table 6.2 shows the impact of altering the values of parameter H, which corresponds to the number of hidden layers and nodes in each layer. While the rest of the parameters remained unchanged as listed in table 4.5.

However, it can be observed that the highest accuracy was achieved on the testing data when we selected the predefined parameter "t" denoting a single hidden layer with 10 nodes. Therefore, the value of parameter H was set to "t" prior to conducting the following parameter tuning of the learning rate.

6.4.3 Learning Rate

Table 6. 3: Performance of different learning rate values

Parameter values	Results of training and validation using 10-fold cross validation	Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4
Default Learning rate 0.3	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 495 – 82.5% Incorrectly classified instances 105 – 17.5%
Learning rate 0.1	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 553 - 92.16% Incorrectly classified instances 47 - 7.83%
Learning rate 0.2	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 493 – 82.16% Incorrectly classified instances 107 – 17.88%
Learning rate 0.4	Correctly Classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 508 – 84.66% Incorrectly classified instances 92 – 15.33%
Learning rate 0.42	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 493 – 82.16% Incorrectly classified instances 107 – 17.83%

6.4.3.1 Discussion

After setting the optimal parameter values of H, we proceed to determine the optimum value of the learning rate L. Table 6.3 displays the effect of varying the values of the learning rate from 0.1 to 0.5 with a step of 0.1. A decrease in the accuracy can be observed when the value exceeds 0.4. While the best result was obtained for a learning rate of 0.1 on testing data as compared to the default value of 0.3.

Therefore, the value of parameter H was set to "t" and the optimal value of L was set to 0.1, prior to conducting the following parameter tuning of the

Momentum. While the rest of the parameters remained unchanged as listed in table 4.5.

6.4.4 Momentum

Table 6. 4: Performance of different momentum values

Parameter values	Results of training and validation using 10-fold cross validation	Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4
Default Momentum 0.2	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 553 – 92.16% Incorrectly classified instances 47 – 7.83%
Momentum 0.1	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 553 – 92.16% Incorrectly classified instances 47 – 7.83%
Momentum 0.3	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 553 – 92.16% Incorrectly classified instances 47 – 7.83%
Momentum 0.4	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 556 – 92.66% Incorrectly classified instances 44 – 7.33%
Momentum 0.5	Correctly classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly classified instances 549 – 91.5% Incorrectly classified instances 51 – 8.5%

6.4.4.1 Discussion

After setting the optimal parameter values of H and L, we proceed to identify the optimum value of the momentum M, where the value of the momentum was varied from 0.1 to 0.5 with an increment of 0.1, and the results of these variations are presented in table 6.4. The same results were obtained when the value changed from 0.1 to 0.3, while the best result was achieved for a

momentum of 0.4 on testing data where a reduction in accuracy was observed when the value of the momentum exceeded 0.4.

In conclusion, better results on testing data were achieved by the MLP classification model with the optimal hyperparameter values of H set to "t", L of 0.1 and M of 0.4 as compared to the original setup using the default values. Where the classification accuracy improved by almost 11.66% and the number of misclassified instances reduced by 11.67%

6.5 SVM Settings

6.5.1 Default Setting

To identify the optimal hyperparameter values of an SVM, the following most common hyperparameters are considered based on [240, 250]: regularisation hyperparameter (-C) and the exponent (-E) value or degree of the kernel.

In WEKA, the default value of the regularisation hyperparameter C is 1.0. while the exponent E value or degree of the selected normalised polynomial kernel by default is 1.0, which behaves like a linear kernel.

Table 6.5 shows classification results obtained using the default parameter settings of SVM in WEKA listed in table 4.5.

Table 6. 5: SVM default setting

Results of training and validation using 10-fold cross validation	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Correctly classified instances 175 - 94%. Incorrectly classified instances 10 - 5.40%.	Correctly Classified instances 482 - 80.33%. Incorrectly classified instances 118 - 19.66%.

6.5.2 Tuning the Values of C and E

In the subsection, the optimal hyperparameter values for C and E are manually searched by adopting the method and range of values used in [240]. While the rest of the parameters remained unchanged as listed in table 4.5.

To determine the optimal values of these hyperparameters, the normalised polynomial kernel was tested using two different exponent values; 1.0, which behaves like a linear kernel, and an exponent or degree value of 2.0 that equivalent to a nonlinear kernel. Whereas the regularisation parameter C was tested over a range of one to four. The effect of varying the values of regularisation parameter C and kernel exponent is shown in table 6.6.

Table 6. 6: Performance for different C and E values

Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Regularisation parameter C = 2 Exponent value = 1.0	Correctly classified instances 177 – 95.67% Incorrectly classified instances 8 – 4.32%	Correctly classified instances 504 - 84% Incorrectly classified instances 96 - 16%
Regularisation parameter C = 3 Exponent value = 1.0	Correctly Classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly Classified instances 510 - 85% Incorrectly classified instances 90 - 15%
Regularisation parameter C = 4 Exponent value = 1.0	Correctly Classified instances 179 – 96.75% Incorrectly classified instances 6 – 3.24%	Correctly Classified instances 510 - 85% Incorrectly classified instances 90 - 15%
Regularisation parameter C = 1 Exponent value = 2.0	Correctly Classified instances 177 – 95.67% Incorrectly classified instances 8 – 4.32%	Correctly Classified instances 525 – 87. 5% Incorrectly classified instances 75 - 12.5%
Regularisation parameter C = 2 Exponent value = 2.0	Correctly Classified instances 179 - 96.75% Incorrectly classified instances 6 - 3.24%	Correctly Classified instances 533 – 88.83% Incorrectly classified instances 67 – 11.16%

Regularisation parameter C = 3 Exponent value = 2.0	Correctly Classified instances 179 - 96.75% Incorrectly classified instances 6 - 3.24%	Correctly Classified instances 536 – 89.33% Incorrectly classified instances 64 – 10.66%
Regularisation parameter C = 4 Exponent value = 2.0	Correctly Classified instances 179 - 96.75% Incorrectly classified instances 6 - 3.24%	Correctly Classified instances 536 – 89.33% Incorrectly classified instances 64 – 10.66%

6.5.3 Discussion

Firstly, in the case when the exponent value E was set to 1, the highest results on testing data were obtained when the value of the regularisation parameter C value was set to 3, while an increment in the C value to 4 caused no change in the model's performance.

Moreover, an improvement in the results was observed when the value of E was set to 2 and the model's performance kept improving as the value of C kept incrementing by 1, while the best performance was achieved when the value of C was set to 3. However, further increasing the value of C to 4 caused no change in the model's performance.

In conclusion, compared to the results presents in table 6.5 for the SVM classification model with the original setup using the default values. Improved results were obtained on the testing data with the optimal hyperparameter values of E = 2 and C = 3. Where the classification accuracy improved by 9% and the number of misclassified instances was also reduced by 9%.

6.6 KNN Settings

6.6.1 Default Setting

As outlined 3.6.3 the value of hyperparameter (-K) number of neighbours, is considered an important hyperparameter that plays a crucial role in the KNN algorithm. The default value of the hyperparameter K in WEKA is 1. Additionally, the default distance measure function is Euclidean distance with

no weight assigned. Table 6.7 shows classification results obtained using the default parameter settings of the KNN in WEKA listed in table 4.5.

Table 6. 7: KNN default setting

Results of training and validation using external 10-fold cross validation	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Correctly Classified instances 179 – 96.75%. Incorrectly classified instances 6 – 3.24%.	Correctly Classified instances 592 – 98.66%. Incorrectly classified instances 8 – 1.33%.

6.6.2 Performing CV Parameter Selection

In this subsection, the optimal value of hyperparameter K is determined by performing the CV Parameter Selection using Euclidean and Manhattan distance functions with different distance weightings. While the rest of the parameters remained unchanged as listed in table 4.5.

This is carried out by following the method of applying different distance functions with different distance weightings and the range of the K value used in [250]. The K value ranged from 1.0 to 10.0 with 10 steps, table 6.8 shows the classification results of the optimal K value for the Euclidean and Manhattan distance functions with different distance weightings. While the rest of the parameters remained unchanged as listed in table 4.5.

Table 6. 8: Performance for the optimal K value using Euclidean and Manhattan distance functions

Parameter values	**Results of training and validation using internal 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Distance function: Euclidean Distance weighting by 1 / distance Optimal K = 3	Correctly classified instances 185 - 100% Incorrectly classified instances 0 – 0%	Correctly classified instances 592 – 98.66% Incorrectly classified instances 8 – 1.33%
Distance function: Euclidean	Correctly classified instances 180 – 97.29%	Correctly classified instances 592 – 98.66%

Distance weighting by 1 – distance Optimal K = 4	Incorrectly classified instances 5 – 2.70%	Incorrectly classified instances 8 – 1.33%
Distance function: Manhattan **Distance weighting by 1 / distance** **Optimal K = 3**	Correctly classified instances 185 - 100% Incorrectly classified instances 0 – 0%	Correctly classified instances 595 – 99.16% Incorrectly classified instances 5 – 0.83%
Distance function: Manhattan Distance weighting by 1 - distance Optimal K = 3	Correctly classified instances 181 – 97.83% Incorrectly classified instances 4 – 2.16%	Correctly classified instances 505 – 89.53% Incorrectly classified instances 59 – 10.46%

6.6.3 Discussion

Based on the classification results on testing data, it can be observed that the highest classification results were obtained with the CV parameter selection returning the optimal K value = 3 using the Manhattan distance function and the distance weighting of 1 / distance. While the same results were obtained on testing data using the Euclidean distance function regardless of the distance weighting used.

Furthermore, compared to the results shown in table 6.7 of KNN with the original setup using the default values. The optimal KNN configuration produced results with 0.5% higher classification accuracy and reduced the number of misclassified instances by 0.5%.

6.7 Decision Tree (C4.5) Settings

6.7.1 Default Setting

To identify the optimal hyperparameter values of decision tree (C4.5), the following most common hyperparameters are considered based on [252]: confidence factor (-C) and the minimum number of instances (minNumObj or -M) in leaf node.

In WEKA the default parameter setting of the C that controls the pruning of the tree is 0.25. Whereas the minimum number of instances M that must be present in a leaf node by default is 2, denoting no further split is carried out if the node contains less than two instances. Table 6.9 shows classification results obtained using the default parameter settings of the decision tree (C4.5) in WEKA listed in table 4.5.

Table 6. 9: Decision tree (C4.5) default setting

Results of training and validation using external 10-fold cross validation	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Correctly classified instances 178 - 96.21%. Incorrectly classified instances 7 - 3.78%.	Correctly classified instances 465 - 77.5%. Incorrectly classified instances 135 - 22.5%.

6.7.2 Performing CV Parameter Selection

In this subsection, the search for the optimal values of hyperparameters M and C was conducted using WEKA's CV Parameter Selection. While the rest of the parameters remained unchanged as listed in table 4.5.

This was carried out by following the method and the range of hyperparameter values used in [252]. Where M ranged from 1.0 to 10.0 with 10 steps, and C ranged from 0.1 to 0.9 with an increment of 0.1. By performing the CV Parameter Selection, the optimal set of parameter values was obtained which is C = 0.5 and M = 1.

Table 6. 10: Performance for the optimal C and M values

Parameter values	**Results of training and validation using internal 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Optimal values C = 0.5 M = 1	Correctly classified instances 185 - 100% Incorrectly classified instances 0 – 0%	Correctly classified instances 473 - 78.83% Incorrectly classified instances 127 - 21.16%

6.7.3 Discussion

Table 6.10 shows the results of the decision tree classification model using the optimal C and M values. Compared to the results obtained using the default configuration in table 6.9, improved results were obtained on the testing data with the optimal hyperparameter values of C = 0.5 and M = 1. Where the classification accuracy of the model was improved by approximately 1.33%, while the number of misclassified instances was reduced by 1.34%.

6.8 Random Forest Settings

6.8.1 Default Setting

To identify the optimal hyperparameter values of Random forest, the following hyperparameters are considered based on [253]: the number of trees that can be generated (-I), the number of features to consider in each split point (-K) and the maximum depth (-depth) of trees indicates how deep the tree would be.

In WEKA, the default settings for these parameters are as follows: the number I is set to 100 by default, the number K is calculated as log2(number of features), and the maximum depth is set to 0 by default, allowing for unlimited depth. Table 6.11 shows classification results obtained using the default parameter settings of the Random forest in WEKA listed in table 4.5.

Table 6. 11: Random forest default setting

Results of training and validation using external 10-fold cross validation	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Correctly classified instances 181 - 97.83%. Incorrectly classified instances 4 - 2.16%.	Correctly classified instances 470 - 78.33%. Incorrectly classified instances 130 - 21.66%.

6.8.2 Performing CV Parameter Selection

Similar to KNN and decision tree, in this subsection the optimal hyperparameter values of I, depth and K are searched by utilising the CV Parameter Selection. While the rest of the parameters remained unchanged as listed in table 4.5.

Where the depth ranged from 1.0 to 10.0 with 10.0 steps, K ranged from 2.0 to 6.0 with 5.0 steps, according to our case with six features and I ranged from 10.0 to 100.0 with 10.0 steps. After performing the CV Parameter Selection, the optimal hyperparameter values obtained were depth = 3, K = 3, and I = 20.

Table 6. 12: Performance of Random forest with optimal configuration

Parameter values	**Results of training and validation using internal 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.4**
Optimal values depth = 3 K = 3 I = 20	Correctly classified instances 185 - 100% Incorrectly classified instances 0 – 0%	Correctly classified instances 470 - 78.33% Incorrectly classified instances 130 - 21.66%

6.8.3 Discussion

The performance of Random forest on testing data using the optimal values of depth = 3, K = 3 and I = 20 is shown in table 6.12. Compared to the results obtained in table 6.11 with the original setup using the default values, no improvement was observed in terms of the classification results. However, the model with the optimal configuration required less computational cost to achieve the same classification results, where a smaller number of trees I = 20 were used as compared to the original setup with the default value of I = 100.

6.9 Repeating the Experiments Using the Optimal Settings

This section repeats the four experiments conducted in chapter five, specifically in sections 5.2 to 5.5. In those experiments, the selected ML classifiers were trained as baseline models using the default hyperparameter settings listed in table 4.5. However, in this section, the same experiments are repeated using the optimal sets of hyperparameter values obtained in sections 6.4 to 6.8 for the five classification models. The remaining parameters, as listed in table 4.5, remain unchanged.

6.9.1 Experimental Setup

For the first experiment, the five ML classifiers are trained on the training data of applications listed in table 5.1 using the optimal sets of hyperparameter values that were obtained in sections 6.4 to 6.8. And their performance is assessed on the testing data of applications listed in table 5.2. While the rest of the parameters remained unchanged as listed in table 4.5.

For the second experiment, the five ML classifiers are trained on the training data of applications listed in table 5.3 using the optimal sets of hyperparameter values that were obtained in sections 6.4 to 6.8. And their performance is assessed on the testing data of applications listed in table 5.4. While the rest of the parameters remained unchanged as listed in table 4.5.

For the third experiment, the five ML classifiers are trained on reduced training data of applications listed in table 5.5, where the size of this training data was reduced to half of the training data used in the second experiment. This is carried out using the optimal sets of hyperparameter values that were obtained in sections 6.4 to 6.8. 4.5. The performance of the classification models is then assessed on the testing data of applications listed in Table 5.4, While the rest of the parameters remained unchanged as listed in table 4.5.

For the fourth experiment, the five ML classifiers are trained on the training data of applications listed in table 5.6 using the optimal sets of hyperparameter values that were obtained in sections 6.4 to 6.8. And their performance is assessed on the testing data of applications listed in table 5.7. While the rest of the parameters remained unchanged as listed in table 4.5.

Moreover, the performance of each classifier is evaluated in terms of classification accuracy, macro-average of precision, recall and weighted average f-measure. Additionally, a confusion matrix is provided to examine the distribution of correct and incorrect predictions made by the classifiers.

6.9.2 Results of the First Experiment: Training with an App of Each Class and Testing on Different App(s) of the Same Class

Classification model: MLP

=== Results ===

Correctly Classified Instances	434	57.86%
Incorrectly Classified Instances	316	42.13%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	138	179	121	12
	Varied	1	148	0	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.796.
Recall: 0.579.
Weighted Avg:
F-Measure: 0.546.

Classification model: KNN

=== Results ===

Correctly Classified Instances	442	58.93%
Incorrectly Classified Instances	308	41.06%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	150	21	238	41
	Varied	0	149	0	1
	Low	0	0	143	7
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.850.
Recall: 0.589.
Weighted Avg:
F-Measure: 0.594.

Classification model: SVM

=== Results ===

Correctly Classified Instances	590	78.66%
Incorrectly Classified Instances	160	21.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	297	144	8	1
	Varied	3	145	2	0
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.881.
Recall: 0.787.
Weighted Avg:
F-Measure: 0.799.

Classification model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	371	49.46%
Incorrectly Classified Instances	379	50.53%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	144	0	201	105
	Varied	1	79	69	1
	Low	0	0	148	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.867.
Recall: 0.495.
Weighted Avg:
F-Measure: 0.533.

Classification model: Random Forest

=== Results ===

Correctly Classified Instances	430	57.33%
Incorrectly Classified Instances	320	42.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	147	5	192	106
	Varied	0	141	9	0
	Low	0	0	142	8
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.876.
Recall: 0.573.
Weighted Avg:
F-Measure: 0.601.

6.9.2.1 Discussion

Compared to the results obtained using WEKA's default configurations in section 5.2, it can be observed that SVM and Random forest produced better results using optimal hyperparameter values, where the classification accuracy of SVM improved by 7.20% and by 2.27% for the Random Forest. Moreover, better results were attained for the macro average of precision, recall, and f-measure when using optimal hyperparameter values for SVM. Similarly, an improvement in the macro average of recall was observed for Random forest, while the macro average of precision and weighted average of f-measure remained slightly better with the WEKA's default configurations. In contrast, overall results for MLP and KNN, including the macro average of precision, recall, and weighted average F-measure, remained better with default hyperparameter settings. For MLP, using the optimal set of hyperparameter values resulted in a decrease in classification accuracy by 1.47%, while for KNN it decreased by 9.87%. Finally, the decision tree classification model yielded the same results as those obtained using the default configuration.

6.9.3 Results of the Second Experiment: Extending the Training Data by Including the Skype Voice Call Application

Classification model: MLP

=== Results ===

Correctly Classified Instances	590	98.33%
Incorrectly Classified Instances	10	1.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	294	2	0	4
	Varied	1	148	0	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.993.
Recall: 0.983.
Weighted Avg:
F-Measure: 0.988.

Classification model: KNN

=== Results ===

Correctly Classified Instances	592	98.66%
Incorrectly Classified Instances	8	1.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	0	149	0	1
	Low	0	0	143	7
	Buffer	0	0	0	0

Macro Avg:
Precision: 1.000.
Recall: 0.987.
Weighted Avg:
F-Measure: 0.993.

Classification model: SVM

=== Results ===

Correctly Classified Instances	566	94.33%
Incorrectly Classified Instances	34	5.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	274	18	7	1
	Varied	4	144	2	0
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.949.
Recall: 0.943.
Weighted Avg:
F-Measure: 0.945.

Classification model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	494	82.33%
Incorrectly Classified Instances	106	17.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	298	0	0	2
	Varied	1	79	69	1
	Low	0	0	117	33
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.906.
Recall: 0.823.
Weighted Avg:
F-Measure: 0.844.

Classification model: Random Forest

=== Results ===

Correctly Classified Instances	578	96.33%
Incorrectly Classified Instances	22	3.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	298	1	1	0
	Varied	0	131	19	0
	Low	0	1	149	0
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.967.
Recall: 0.963.
Weighted Avg:
F-Measure: 0.963.

6.9.3.1 Discussion

Compared to the experimental results obtained using WEKA's default settings in section 5.3, slightly better results, including for the macro average of precision, recall, and weighted average f-measure, were achieved by classification models on testing data using the optimal set of hyperparameter values. However, no improvement was observed in the performance of the decision tree classification model, as the results obtained using the optimal set of hyperparameter values were identical to those obtained using WEKA's default setting.

6.9.4 Results of the Third Experiment: Reducing the Training Data by Half

Classification model: MLP

=== Results ===

Correctly Classified Instances	579	96.5%
Incorrectly Classified Instances	21	3.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	284	0	0	16
	Varied	1	148	0	1
	Low	0	0	147	3
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.998.
Recall: 0.965.
Weighted Avg:
F-Measure: 0.981.

Classification model: KNN

=== Results ===

Correctly Classified Instances	563	93.83%
Incorrectly Classified Instances	37	6.16%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	0	149	0	1
	Low	0	0	114	36
	Buffer	0	0	0	0

Macro Avg:
Precision: 1.000.
Recall: 0.938.
Weighted Avg:
F-Measure: 0.965.

Classification model: SVM

=== Results ===

Correctly Classified Instances	564	94%
Incorrectly Classified Instances	36	6%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	274	22	0	4
	Varied	5	142	2	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.953.
Recall: 0.940.
Weighted Avg:
F-Measure: 0.945.

Classification model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	426	71%
Incorrectly Classified Instances	174	29%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	0	1	0
	Varied	0	41	107	2
	Low	63	0	86	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.774.
Recall: 0.710.
Weighted Avg:
F-Measure: 0.684.

Classification model: Random Forest

=== Results ===

Correctly Classified Instances	526	87.66%
Incorrectly Classified Instances	74	12.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	0	1	0
	Varied	0	113	36	1
	Low	31	0	114	5
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.892.
Recall: 0.877.
Weighted Avg:
F-Measure: 0.879.

6.9.4.1 Discussion

The results of this experiment showed an improvement in the performance of MLP and Random forest using the optimal set of hyperparameter values when compared to the results obtained in section 5.4 using WEKA's default hyperparameter settings. While a drop of only 0.17% in the classification accuracy of the KNN model was observed with the optimal set of hyperparameter values. Moreover, a slight improvement was observed in the performance of SVM and decision tree classification models using the optimal set of hyperparameter values, where the classification accuracy was improved by 2.34% for the SVM and only by 0.17% for the decision tree. Overall improvements in the results, including for the macro average of precision, recall, and weighted average f-measure were observed by using the optimised hyperparameters.

6.9.5 Results of the Fourth Experiment: Further Assessment of the Generalisation Capacity

Classification model: MLP

=== Results ===

Correctly Classified Instances	587	97.83%
Incorrectly Classified Instances	13	2.16%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	294	6	0	0
	Varied	0	148	0	2
	Low	1	2	145	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.985.
Recall: 0.978.
Weighted Avg:
F-Measure: 0.982.

Classification model: KNN

=== Results ===

Correctly Classified Instances	589	98.16%
Incorrectly Classified Instances	11	1.83%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	298	0	0	2
	Varied	1	147	0	2
	Low	1	2	144	3
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.993.
Recall: 0.982.
Weighted Avg:
F-Measure: 0.987.

Classification model: SVM

=== Results ===

Correctly Classified Instances	473	78.83%
Incorrectly Classified Instances	127	21.16%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	195	105	0	0
	Varied	12	133	0	5
	Low	1	1	145	3
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.858.
Recall: 0.788.
Weighted Avg:
F-Measure: 0.801.

Classification model: Decision tree (C4.5)

=== Results ===

Correctly Classified Instances	597	99.5%
Incorrectly Classified Instances	3	0.5%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	1	0	0
	Varied	0	150	0	0
	Low	1	1	148	0
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.995.
Recall: 0.995.
Weighted Avg:
F-Measure: 0.995.

Classification model: Random Forest

=== Results ===

Correctly Classified Instances	594	99%
Incorrectly Classified Instances	6	1%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	299	1	0	0
	Varied	0	150	0	0
	Low	0	2	145	3
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.995.
Recall: 0.990.
Weighted Avg:
F-Measure: 0.992.

6.9.5.1 Discussion

Compared to the results obtained in section 5.5 using WEKA's default hyperparameter settings, the only improvement observed by using the optimal set of hyperparameter values was in the performance of MLP, where the classification accuracy was only improved by 0.17. Both KNN and decision tree classification models produced the same results with the optimal hyperparameters as those obtained using WEKA's default configurations. These results were also similar in terms of the macro average of precision, recall, and weighted average f-measure. Moreover, the results achieved by SVM and Random forest using WEKA's default hyperparameter settings remained better, even in terms of the weighted average of precision, recall, and

f-measure, than those obtained using the optimal set of hyperparameter values. With the optimal set of hyperparameter values, the classification accuracy of SVM dropped from 93% to 78.83% and from 99.16% to 99% for the random forest.

6.9.6 Conclusion

In this section, the previous experiments conducted in chapter five sections 5.2 to 5.5 were repeated to assess the performance of the ML classification models using the optimal sets of hyperparameter values obtained in sections 6.4 to 6.8.

However, the results of these experiments showed that using the optimal set of hyperparameter values did not always lead to better results compared to those obtained with default configurations. In some cases, the results obtained using WEKA's default hyperparameter settings were similar to or better than those achieved using the optimal set of hyperparameter values.

Since the optimal sets of hyperparameter values for classification models were determined using the reduced training data to a quarter of the size in experiment three section 5.4 consisting of 185 samples. Therefore, overall improvements in the results were observed by using the optimised hyperparameters on the full training data of 750 samples in the second experiment section 5.3 and on half of the size in the third experiment section 5.4. This is because the training data being used in the second and third experiments is the same and the only difference is in the size of the training data.

However, in the first and fourth experiments (sections 5.2 and 5.5), where the overall distribution of the training data was different from that used in the second and third experiments, the default hyperparameter settings in some cases performed comparably or better than the optimised hyperparameters.

This indicates that optimising hyperparameters for a particular training data may not always improve the model's performance when the overall distribution of the training data changes, and default settings may be just as effective or even better.

6.10 Optimal Hyperparameter Settings for the First and the Fourth Experiments

Since the experimental results in the previous section, particularly of the repeated experiments one and four indicated that using the optimised hyperparameters for a particular training data may not always lead to improved model performance when there are changes in the overall distribution of new training data.

6.10.1 Experimental Setup

Therefore, in this section, the optimal sets of hyperparameter values for classification models in the first and fourth experiments of (sections 5.2 and 5.5) are determined. The same method for identifying the optimal sets of hyperparameter values described in 6.3 and employed in sections 6.4 to 6.8 was applied to determine the optimal settings that result in better-performing classification models. Where 10-fold cross-validation is performed to explore different hyperparameter settings on the training data of the applications listed in table 5.1 of the first experiment and table 5.6 of the fourth experiment. This is followed by evaluating the performance of the constructed classification models on the testing data of the applications listed in table 5.2 of the first experiment and table 5.7 of the fourth experiment.

6.10.2 Results of the First Experiment

Classification model: MLP

Table 6. 13: Performance of MLP

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 584 – 97.33%. Incorrectly classified instances 16 – 2.66%.	Correctly classified instances 445 – 59.33%. Incorrectly classified instances 305 – 40.66%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Optimal values - H = 10,10. Two hidden layers with 10 nodes in each hidden layer. - L = 0.1. - M = 0.4. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 590 – 98.33%. Incorrectly classified instances 10 – 1.66%.	Correctly classified instances 447 – 59.6%. Incorrectly classified instances 303 – 40.4%.

Classification model: SVM

Table 6. 14: Performance of SVM

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 574 – 95.66%. Incorrectly classified instances 26 – 4.33%.	Correctly classified instances 536 – 71.46%. Incorrectly classified instances 214 – 28.53%.

Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Optimal values - Regularisation parameter C = 2. - Exponent value = 2.0. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 579 – 96.5% Incorrectly classified instances 21 – 3.5%	Correctly classified instances 612 – 81.6% Incorrectly classified instances 138 – 18.4%

Classification model: KNN

Table 6. 15: Performance of KNN

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 593 – 98.83%. Incorrectly classified instances 7 – 1.16%.	Correctly classified instances 516 – 68.8%. Incorrectly classified instances 234 – 31.2%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Optimal values - Distance function: Euclidean. - Distance weighting: with no distance weight assigned. Optimal K = 2. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 596 – 99.33%. Incorrectly classified instances 4 – 0.66%.	Correctly classified instances 519 – 69.2%. Incorrectly classified instances 231 – 30.8%.

Classification model: Decision tree (C4.5)

Table 6. 16: Performance of Decision tree (C4.5)

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Using default parameter settings listed in table 4.5.	Correctly classified instances 590 – 98.33%. Incorrectly classified instances 10 – 1.66%.	Correctly classified instances 371 – 49.46%. Incorrectly classified instances 379 – 50.53%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Optimal values - C = 0.3. - M = 4. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 594 - 99% Incorrectly classified instances 6 – 1%	Correctly classified instances 374 – 49.86% Incorrectly classified instances 376 – 50.13%

Classification model: Random forest

Table 6. 17: Performance of Random forest

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Using default parameter settings listed in table 4.5.	Correctly classified instances 594 - 99% Incorrectly classified instances 6 - 1%	Correctly classified instances 413 – 55.06% Incorrectly classified instances 337 – 44.93%

Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.2**
Optimal values - depth = 5. - K = 2. - I = 50. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 600 – 100% Incorrectly classified instances 0– 0%	Correctly classified instances 492 – 65.6% Incorrectly classified instances 258 – 34.4%

6.10.2.1 Discussion

Tables 6.13 to 6.17 show the results obtained from classification models using both the default and optimised hyperparameter settings. It can be observed that all classification models achieved better results on testing data using the optimal sets of hyperparameter values as compared to the results obtained using the default hyperparameter settings. Where classification accuracy of Random forest and SVM improved by 10.54% and 10.14% respectively, while it improved by 0.27% for MLP. Finally, both KNN and the decision tree showed an improvement of 0.4%.

6.10.3 Results of the Fourth Experiment

Classification model: MLP

Table 6. 18: Performance of MLP

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 736 – 98.13%. Incorrectly classified instances 14 – 1.86%.	Correctly classified instances 586 – 97.66%. Incorrectly classified instances 14 – 2.33%.

Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Optimal values - H = "i" = 6 nodes in a single hidden layer. - L = 0.1. - M = 0.2. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 740 – 98.66%. Incorrectly classified instances 10 – 1.33%.	Correctly classified instances 588 – 98%. Incorrectly classified instances 12 – 2%.

Classification model: SVM

Table 6. 19: Performance of SVM

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 716 – 95.46%. Incorrectly classified instances 34 – 4.53%.	Correctly classified instances 558 – 93%. Incorrectly classified instances 42 – 7%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Optimal values - Regularisation parameter C = 1.4. - Exponent value = 1.0. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 731 – 97.46%. Incorrectly classified instances 19 – 2.53%.	Correctly classified instances 564 – 94%. Incorrectly classified instances 36 – 6%.

Classification model: KNN

Table 6. 20: Performance of KNN

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Using default parameter settings listed in table 4.5.	Correctly classified Instances 743 – 99.06%. Incorrectly classified instances 7 – 0.93%.	Correctly classified instances 589 – 98.16%. Incorrectly classified instances 11 – 1.83%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Optimal values - Distance function: Manhattan. - Distance weighting by 1 / distance. - Optimal K = 2. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 750 – 100%. Incorrectly classified instances 0 – 0%.	Correctly classified instances 592 – 98.66%. Incorrectly classified instances 8 – 1.33%.

Classification model: Decision tree (C4.5)

Table 6. 21: Performance of Decision tree (C4.5)

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Using default parameter settings listed in table 4.5.	Correctly classified instances 748 – 99.73%. Incorrectly classified instances 2 – 0.26%.	Correctly classified instances 597 – 99.5%. Incorrectly classified instances 3 – 0.5%.

Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Optimal values - C = 0.6. - M = 1. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 750 - 100%. Incorrectly classified instances 0– 0%.	Correctly classified instances 597 – 99.5%. Incorrectly classified instances 3 – 0.5%.

Classification model: Random forest

Table 6. 22: Performance of Random forest

Default hyperparameter settings		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Using default parameter settings listed in table 4.5.	Correctly classified instances 745 – 99.33%. Incorrectly classified instances 5 – 0.66%.	Correctly classified instances 595 – 99.16%. Incorrectly classified instances 5 – 0.83%.
Optimal hyperparameter values		
Parameter values	**Results of training and validation using 10-fold cross validation**	**Results of testing the constructed classification model on unseen testing data of applications listed in table 5.7**
Optimal values - depth = 7. - K = 2. - I = 40. - The rest of the parameters remained unchanged as listed in table 4.5.	Correctly classified instances 750 – 100%. Incorrectly classified instances 0 – 0%.	Correctly classified instances 596 – 99.33%. Incorrectly classified instances 4 – 0.66%.

6.10.3.1 Discussion

The experimental results of using both the default and optimised hyperparameter settings are shown in tables 6.18 to 6.22. Overall improvements in the results were observed by using the optimised hyperparameters. Where the classification accuracy of the MLP improved by 0.34% on testing data compared to the default hyperparameter settings, while it improved by 1% for the SVM classification model. Moreover, an improvement of 0.5% in the accuracy of KNN was observed, while an improvement of 0.17% was observed in the accuracy of the Random Forest. Furthermore, the decision tree model showed an improvement of 0.27% on the training data using cross-validation, although the accuracy remained the same at 99.5% on the testing data.

Overall, the results in this section confirm that better results can be obtained by conducting a hyperparameter optimisation process independently for each training data.

6.11 Further Analyses

This section provides a deeper analysis of the confusion matrix, focusing on the breakdown of the predictions, including the distribution of correct and incorrect predictions made by the classification model.

6.11.1 Confusion Matrix

To analyse the confusion matrix, we consider the confusion matrix of an MLP classification model in subsection 6.9.3 Results of the second experiment: Extending the training data by including the Skype voice call application.

Classification model: MLP

=== Results ===

Correctly Classified Instances	590	98.33%
Incorrectly Classified Instances	10	1.66%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	294	2	0	4
	Varied	1	148	0	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.993.
Recall: 0.983.
Weighted Avg:
F-Measure: 0.988.

Each element in a confusion matrix denotes the number of predictions made by a classification model for a specific class. Moreover, it provides insights into these predictions by indicating whether the predictions were classified correctly or incorrectly.

There are four terms used for interpreting the confusion matrix:

True Positives (TP): where the classifier correctly predicts the positive class, and the actual is positive.

True Negatives (TN): where the classifier correctly predicts the negative class, and the actual is negative.

False Positives (FP): where the classifier incorrectly predicts the positive class, and the actual is negative.

False Negatives (FN): where the classifier incorrectly predicts the negative class, and the actual is positive.

Where the TP, TN, FP and FN for class high are as follows:

TP = 294.

TN = 148 + 0 + 1 + 1 + 148 + 1 + 0 + 0 + 0 = 299.

FP = 1 + 0 + 0 = 1.

FN = 2 + 4 = 6.

And the evaluation metrics for class high are calculated as follows:

Precision = True Positives / (True Positives + False Positives).

= 294 / (294+1) = 0.997.

Recall or sensitivity = True Positives / (True Positives + False Negatives).

= 294 / (294+6) = 0.980.

F-measure = 2 * (precision * recall) / (precision + recall).

= 2 * (0.997 * 0.980) / (0.997 + 0.980) = 0.988.

Specificity = True Negatives / (True Negatives + False Positives).

= 299 / (299+1) = 0.997.

The TP, TN, FP and FN for class Varied are calculated as follows:

TP = 148.

TN = 294 + 0 + 4 + 0 + 148 + 1 + 0 + 0 + 0 = 447.

FP = 2 + 1 + 0 = 3.

FN = 1 + 0 + 1 = 2.

Precision = True Positives / (True Positives + False Positives).

= 148 / (148+3) = 0.980.

Recall or sensitivity = True Positives / (True Positives + False Negatives).

= 148 / (148+2) = 0.987.

F-measure = 2 * (precision * recall) / (precision + recall).

= 2 * (0.980 * 0.987) / (0.980 + 0.987) = 0.983.

Specificity = True Negatives / (True Negatives + False Positives).

= 447 / (447+3) = 0.993.

The TP, TN, FP and FN for class Low are calculated as follows:

TP = 148.

TN = 294 + 2 + 4 + 1 + 148 + 1 + 0 + 0 + 0 = 450.

FP = 0 + 0 + 0 = 0.

FN = 1 + 0 + 1 = 2.

Precision = True Positives / (True Positives + False Positives).

= 148 / (148+0) = 1.000.

Recall or sensitivity = True Positives / (True Positives + False Negatives).

= 148 / (148+2) = 0.987.

F-measure = 2 * (precision * recall) / (precision + recall).

= 2 * (1.000 * 0.987) / (1.000 + 0.987) = 0.993.

Specificity = True Negatives / (True Negatives + False Positives).

= 450 / (450+0) = 1.000.

Moreover, the current model handles the true negative instances that are correctly identified as not belonging to a certain class. For example, if the model correctly predicts that an instance does not belong to the High class, then it must belong to one of the remaining three classes. Since the classifier was not practically trained to consider instances of none of the above classes.

6.11.2 Cost Matrix

This subsection describes the process of reweighting the training inputs by assigning costs to the class misclassifications in the cost matrix.

By default, in WEKA, the output classification threshold set to 0.5, assuming a balanced distribution of class labels, where the classification treats all misclassifications (false positives and false negatives) equally. Cost sensitive classification is a method of reweighting the training inputs based on predefined class cost of misclassification or estimating a class with the lowest misclassification cost. It involves adjusting the probability threshold of the classifier's output based on the cost of misclassifications [256, 257].

Equation 6.1 represents the general cost matrix C; the diagonal elements represent the cost of correct classifications and μ and λ denote the assigned costs of the class misclassifications. Where μ represents the cost of false positives and λ represents the cost of false negatives [256].

$$C = \begin{bmatrix} 0 & \lambda \\ \mu & 0 \end{bmatrix} \tag{6.1}$$

The following experiments are carried out by employing the MLP classification model that was used for analysing the confusion matrix in the previous subsection 6.11.1. We have utilised the same experimental setup of 6.9.1, where the MLP was trained on the training data of applications listed in table 5.3 using the optimal sets of hyperparameter values that were obtained

in section 6.4 and assessed on the testing data of applications listed in table 5.4. While the rest of the parameters remained unchanged as listed in table 4.5.

By examining the confusion matrix in subsection 6.11.1, we can observe that for the class high, the model correctly predicted 294 instances as TP, where the predicted class and the actual class were the same. Moreover, it predicted 299 instances as TN, so its prediction in classifying these instances into other classes than high was correct.

In terms of misclassification (FP and FN) for the class high, the model incorrectly predicted 1 instance as FP, where its prediction to classify this instance as class high was incorrect, as its actual class is varied. Moreover, the model incorrectly predicted 6 instances as FN. So, its prediction in classifying these instances into other classes than high was incorrect since their actual class is high.

Furthermore, in terms of other classes, for class varied, the model correctly predicted 148 instances as TP, it also correctly predicted 447 instances as TN. While it incorrectly predicted 3 instances as FP and 2 instances as FN.

In terms of class low, the model correctly predicted 148 instances as TP, it also correctly predicted 450 instances as TN. While it incorrectly predicted 2 instances as FN and had no FP.

The following cost matrix corresponds to the confusion matrix of section 6.11.1.

Cost Matrix
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

To minimise the misclassifications of class high, as observed in the confusion matrix of subsection 6.11.1, the model has a total of 7 misclassifications for the class high, including 1 FP and 6 FNs. Where the model incorrectly predicted 1 instance as FP and its prediction to classify this instance as class high was incorrect, as its actual class is varied. Additionally, the model incorrectly predicted 6 instances as FN and its prediction to classify these 6 instances into other classes (2 as varied and 4 as buffer) was incorrect, as their actual class is high.

Consequently, in the subsequent experiment, we adjust the misclassification of false negatives. In the corresponding cost matrix, the cost value assigned to 4 FNs is adjusted to 2.

Cost Matrix
0 1 1 2
1 0 1 1
1 1 0 1
1 1 1 0

Classification model: MLP

=== Results ===

Correctly Classified Instances	592	98.66%
Incorrectly Classified Instances	8	1.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	296	1	0	3
	Varied	1	148	0	1
	Low	0	1	148	1
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.995.
Recall: 0.987.
Weighted Avg:
F-Measure: 0.991.

Compared to the confusion matrix in subsection 6.11.1, it can be seen that 2 instances out of the 6 false negatives are now correctly classified as true positives. Additionally, the count of false positives for class varied reduced from 3 to 2 and it reduced from 6 to 5 for class buffer. However, since 4 instances are still incorrectly misclassified as false negatives by the model, the

cost value assigned to false negatives is adjusted to 3 for the following experiment.

Cost Matrix
0 1 1 3
1 0 1 1
1 1 0 1
1 1 1 0

Classification model: MLP

=== Results ===

Correctly Classified Instances	595	99.16%
Incorrectly Classified Instances	5	0.83%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	300	0	0	0
	Varied	1	148	0	1
	Low	0	1	147	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.997.
Recall: 0.992.
Weighted Avg:
F-Measure: 0.994.

Compared to the confusion matrix in subsection 6.11.1, now it can be seen that the model correctly classified all instances as true positives for the class high.

Moreover, the count of false positives for class varied reduced from 3 to 1 and it reduced from 6 to 3 for class buffer. However, in terms of the FNs of the class low, the model incorrectly classified 1 more instance as an FN. The total count of FNs for class low is now 2, whereas it was only 1 in the confusion matrix of section 6.11.

Therefore, the cost value assigned to these two FNs is adjusted to 2 in the cost matrix for the following experiment.

Cost Matrix
0 1 1 3
1 0 1 1
1 1 0 2
1 1 1 0

Classification model: MLP

=== Results ===

Correctly Classified Instances	586	97.66%
Incorrectly Classified Instances	14	2.33%

=== Confusion Matrix ===

		Predicted			
		High	**Varied**	**Low**	**Buffer**
Actual	**High**	291	1	0	8
	Varied	1	148	0	1
	Low	0	1	147	2
	Buffer	0	0	0	0

Macro Avg:
Precision: 0.995.
Recall: 0.977.
Weighted Avg:
F-Measure: 0.986.

By observing the confusion matrix, it can be seen that no further improvement can be achieved, and the model's overall performance can decline after reaching a certain point.

6.11.3 Discussion

The experimental results showed that reweighting the training inputs by assigning costs of the class misclassifications in the cost matrix, allows a classification model to consider the varying costs associated with misclassifications. This enhances the model's predictions with a focus on minimising the overall cost of misclassifications. However, careful consideration should be given to the design of the cost matrix by taking into account the consequences of misclassifications.

Moreover, in a multi class classification, the FNs of one class correspond to the FPs of other classes. Thus, when the number of FNs for a specific class was minimised, this indirectly led to a reduction in the FP for the remaining classes.

6.12 Summary

In this chapter, the process of hyperparameter optimisation was carried out to identify the optimal settings that result in a better-performing classification model. This chapter also reviewed the common hyperparameter tuning

methods. It then described the experimental setup employed in this research to perform the hyperparameter optimisation process. This was followed by conducting the hyperparameter optimisation process using both manual and automated tuning methods. Where this was carried out by performing 10-fold cross-validation on the training data of the applications listed in table 5.5 of experiment three section 5.4 consisting of 185 samples. This chapter also evaluated the performance of the constructed classification models using the obtained optimal sets of hyperparameter values on unseen testing data of the applications listed in table 5.4. This chapter further assessed the performance of the classification models by repeating the previous four experiments conducted in chapter five, using the optimal sets of hyperparameter values that were obtained through the optimisation process. Moreover, since the experimental results of the repeated experiments one and four showed that the default hyperparameter settings, in some cases, performed comparably or better than the optimised hyperparameters. Further hyperparameter tuning was performed in this chapter, where the optimal sets of hyperparameter values were determined for classification models of the first and fourth experiments (sections 5.2 and 5.5). Where this was achieved by performing 10-fold cross-validation and exploring different hyperparameter settings on the training data of the applications listed in table 5.1 of the first experiment and table 5.6 of the fourth experiment. This was followed by evaluating the performance of the constructed classification models on the testing data of the applications listed in table 5.2 of the first experiment and table 5.7 of the fourth experiment.

7

PERFORMANCE EVALUATION OF CALI POWER SAVING MODES

7.1 Introduction

The experiments in this chapter are conducted to observe the effect of adjusting the listen interval on energy consumption after the ML classification model has classified new unseen samples into one of the output modes. Subsection 7.2.1 begins by describing the experimental setup employed in the creation of the corresponding traffic scenarios of CALI power saving modes. Subsection 7.2.2 focuses on assessing the performance of CALI power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, using varied sets of energy parameters. This is followed by assessing the performance of CALI against the value variations of energy parameters in subsection 7.2.3.

7.2 CALI Power Saving Modes

7.2.1 Experimental Setup

The described NS-2 extension in subsection 2.3.2 supports a WLAN in an infrastructure mode, where two wireless devices are connected to an AP. The first wireless device (node 1) sends data destined to wireless device 2 (node 2) via AP (node 0).

This extension also supports power management functions such as PS-Poll, AP buffer, TIM, and listen interval. Additionally, it includes an energy model that supports various energy parameters, enabling the estimation of energy consumption for wireless nodes within a network. However, to conduct this experiment the following steps were followed:

First step

To experiment with the four CALI power saving modes, we used the provided Tcl script in tcl/ex/powersave.tcl. And then created four corresponding traffic scenarios: Buffering, DLI, Low, and Awake.

Second step

To estimate the energy consumption for wireless nodes within a network. We have configured the energy parameters for the energy model using three sets of energy parameters reported in major previous studies. Each set consists of 6 energy parameters; Set 1 has been widely employed in studies including [18, 151, 152]. Set 2 reflects the energy parameters of Wavelan WNIC [153, 1], whereas Set 3 reflects the energy parameters of Intel WNIC [154, 155].

The six parameters are:

1. txPower: the power consumption during packet transmission.
2. rxPower: the power consumption during packet reception.
3. idlePower: the power consumption when a WNIC is awake and not transmitting or receiving packets.
4. transitionPower: the power consumption when a WNIC transits from the sleep to idle state and vice versa. This must be twice that of idlePower [151].
5. transitionTime: The amount of time required when a WNIC transits from sleep to idle state and vice versa.

6. sleepPower: The power consumption when a WNIC is in sleep state.

The three sets of energy parameters are shown in table 7.1.

Table 7. 1: Sets of energy parameters

Parameter	Set 1 Value	Set 2 Value	Set 3 Value
txPower	1.4 W	1.675 W	1.44 W
rxPower	0.9 W	1.425 W	1.34 W
idlePower	0.7 W	1.319 W	1.27 W
transitionPower	1.4 W	2.638 W	2.54 W
transitionTime	0.002 S	0.002 S	0.002 S
sleepPower	0.06 W	0.177 W	0.22 W

Third step

The third step involves configuring the traffic for the corresponding scenarios within the Tcl script. Since smartphone applications spend longer in receiving packets than transmitting, the downlink receiving traffic has been considered in our simulation of node 2. From the dataset, we have used the following features as inputs to configure the four corresponding traffic scenarios of CALI power saving modes: 1- receiving data rates, 2- number of received bytes, and 3- number of received packets.

Moreover, to establish the traffic flow between node 1 and node 2 using UDP through node 0, the UDP connection and the source (Agent/UDP) are defined at node 1 using the following:

```
set udp [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp
```

And the UDP destination (Agent/UDP) is defined at node 2.

set null [new Agent/Null]

$ns_ attach-agent $node_(2) $null

$ns_ connect $udp $null

This is followed by defining the Constant Bit Rate (CBR) traffic generator model that uses UDP to send the traffic.

set cbr [new Application/Traffic/CBR].

Where the buffering scenario uses traffic from the XiiaLive internet radio application using a random station with a 128kbps stream.

For the DLI scenario, the traffic of 30 emails in Gmail and receiving 30 Facebook posts at random intervals was employed.

For the low scenario, NSS was run several times. We observed that the duration of one game is about 110 seconds, after that time an advertisement will be loaded.

Finally, for the awake scenario, traffic of 10min Skype video call was used.

Fourth step

The fourth step involves optimising the sleep and awake cycles of the WNIC, which is achieved by adjusting the listen interval.

Finally, the simulation environment is based on Ubuntu 10.04.4 LTS, which is compatible with the NS-2 extension. The simulation duration is set to 600 seconds and initial energy of 1000 J.

7.2.2 Results and Analysis

This section evaluates the performance of CALI power saving modes by comparing the levels of energy consumption of CALI with existing power

saving approaches. We selected APSM as the most current approach deployed in smartphones and SAPSM as a recent technique also employing ML.

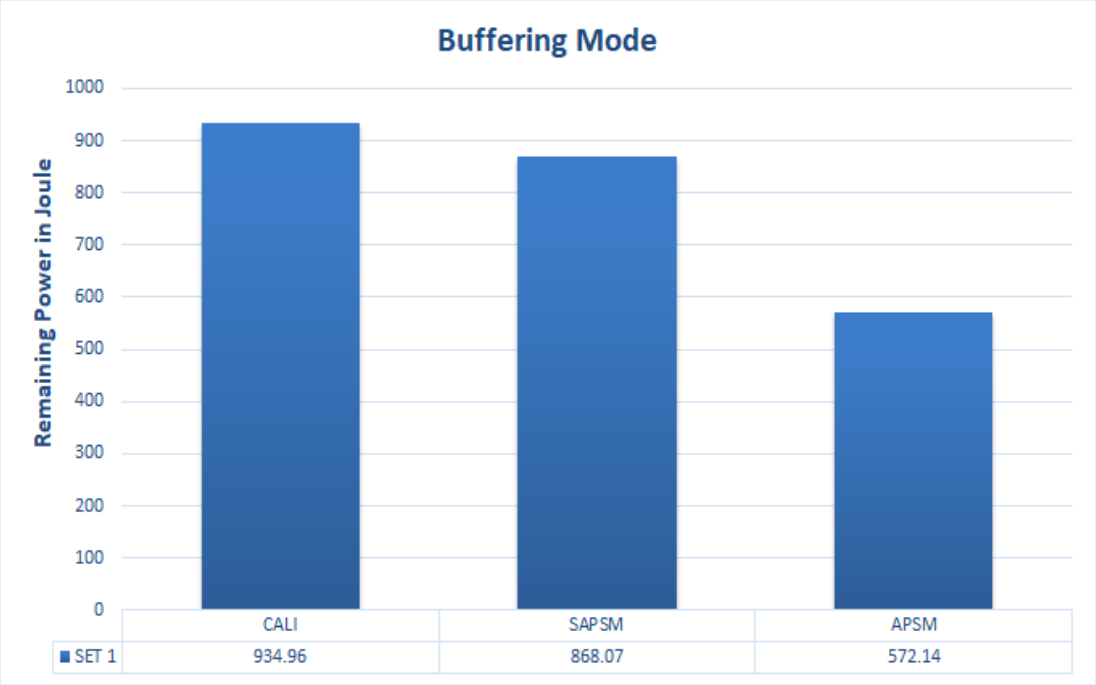


Figure 7. 1: Comparison of CALI, SAPSM, and APSM in buffering mode against set 1 of energy parameters

Figures 7.1, 7.2 and 7.3 show the energy consumption of CALI, SAPSM, and APSM in buffering mode for the 3 sets of energy parameters. We set the listen interval of CALI to 10 = 1000ms. The listen interval value has been determined to not affect audio quality in several experiments with the audio streaming application XiiaLive.

We found that the added delay did not impact the playback streaming quality as was also noted in [10] and [142]. For all 3 sets of energy parameters, CALI consumes less energy in comparison to SAPSM and APSM. In Set 2, CALI consumes 14.14% less energy compared to SAPSM and 75.89% when compared with APSM. For all 3 sets of energy parameters, APSM consumes more energy in comparison to SAPSM and CALI. This is due to the behaviour of APSM with this type of traffic, as the WNIC remains awake and always on.

When the values of rxPower and idlePower increased in Set 2, more power was consumed using APSM compared to Set 1 and Set 3.

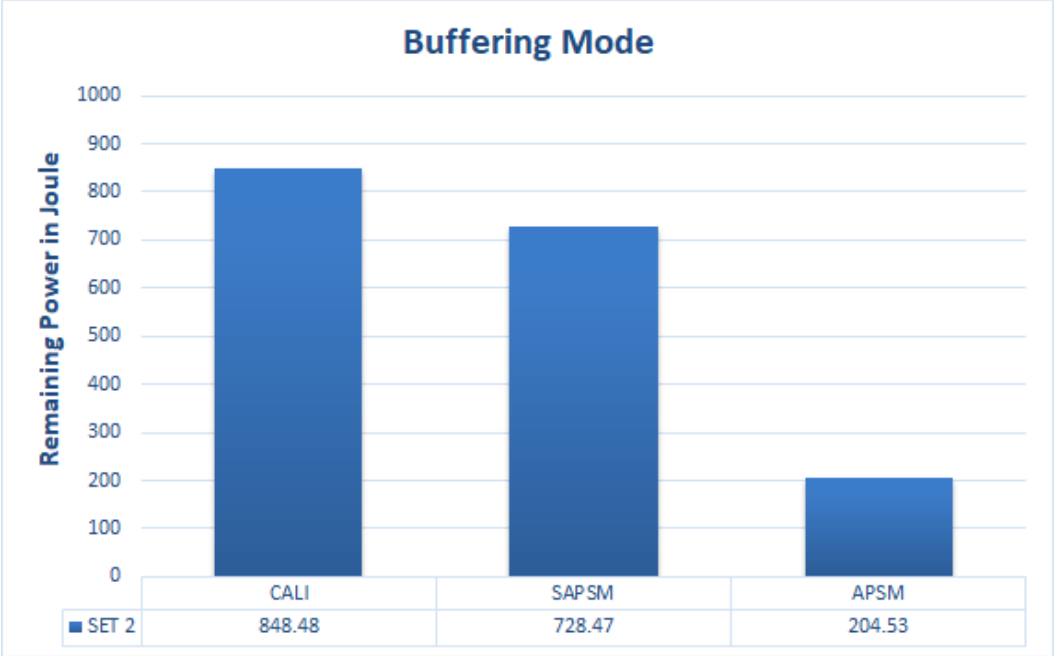


Figure 7. 2: Comparison of CALI, SAPSM, and APSM in buffering mode against set 2 of energy parameters

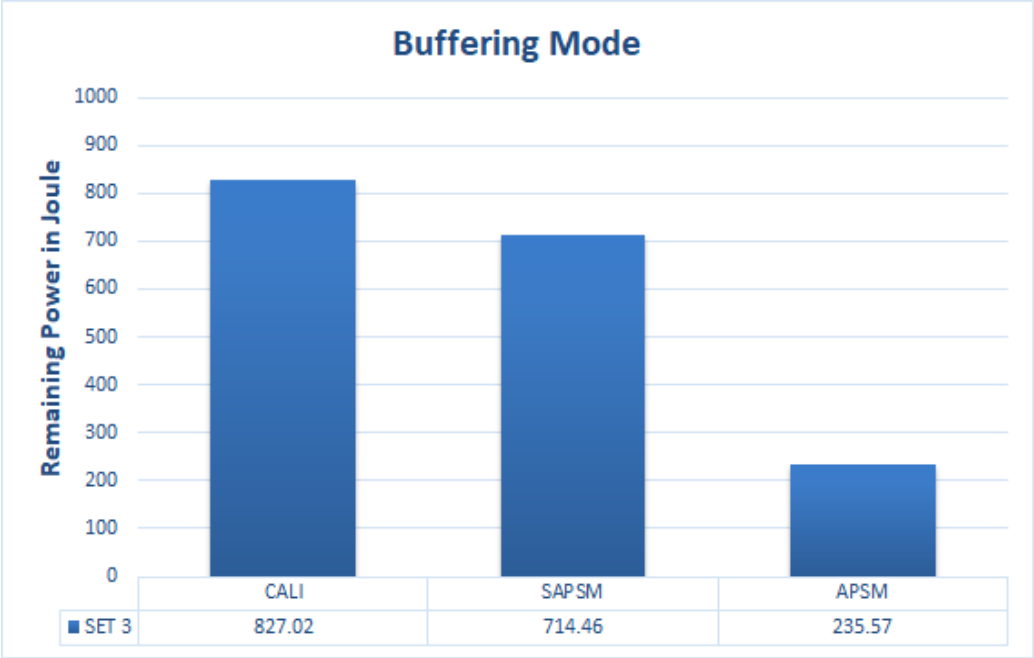


Figure 7. 3: Comparison of CALI, SAPSM, and APSM in buffering mode against set 3 of energy parameters

Figures 7.4, 7.5 and 7.6 show the levels of energy consumption of CALI, SAPSM, and APSM in DLI mode for the 3 sets of energy parameters. Recall that for DLI mode, the listen interval of a wireless device is incremented by 1 at each time a wireless device wakes up during its listen interval and does not find any packets buffered at the AP, and reverting to 1 when interactions occur.

We adjusted the listen interval of CALI to 2,4,6,8, and 10, for applications with varied levels of network activity (Gmail and Facebook), as these applications have intermittent network interactions and do not always receive data. Based on 30 emails and 30 Facebook posts, CALI consumes less energy in comparison to SAPSM and APSM for all 3 sets of energy parameters. Figure 7.5 shows CALI consumes 8.58% to 14.37% less energy compared to SAPSM when the listen interval is set to between 2 and 10. This increases to between 44.48% and 48.00% less energy in comparison with APSM. In contrast, APSM consumes more energy than SAPSM and CALI in all 3 sets of energy parameters. As it switches to awake mode when interaction occurs in the background and remains in awake mode for an idle timeout period before fully switching back to SPSM.

Although these applications run in the background non-interactively and do not always receive data, SPSM could add an approximate delay of 100-300ms of delay when the WNIC is off during the beacon intervals, but buffered packets are available at the AP. This added delay could reach 1000ms in the case of CALI when the listen interval is increased to 10.

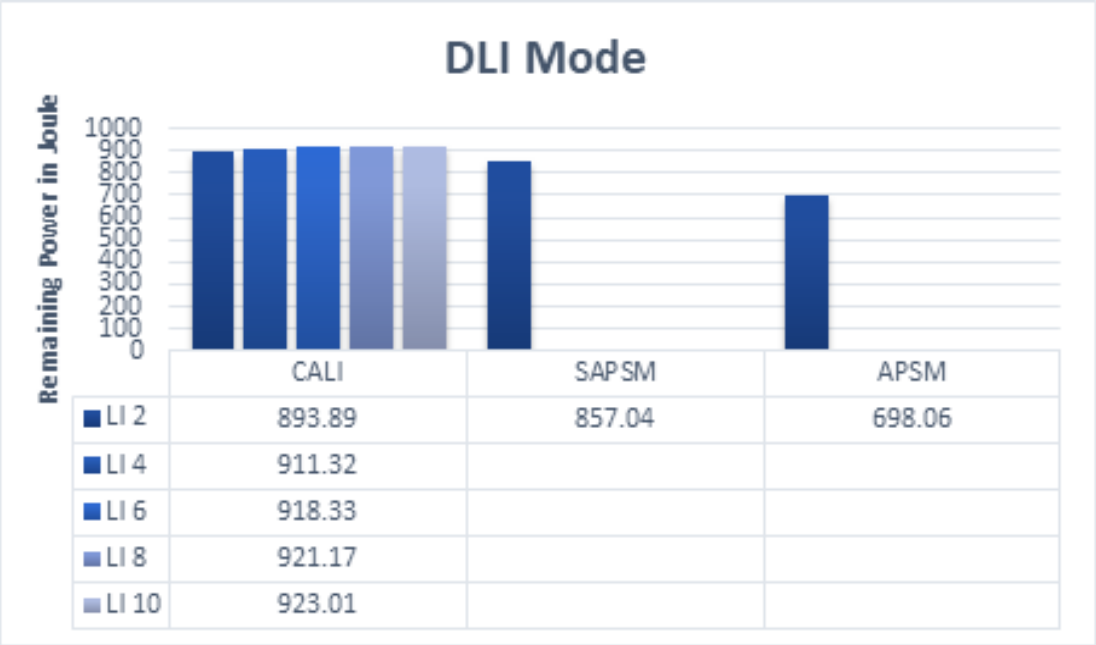


Figure 7. 4: Comparison of CALI, SAPSM, and APSM in DLI mode against set 1 of energy parameters

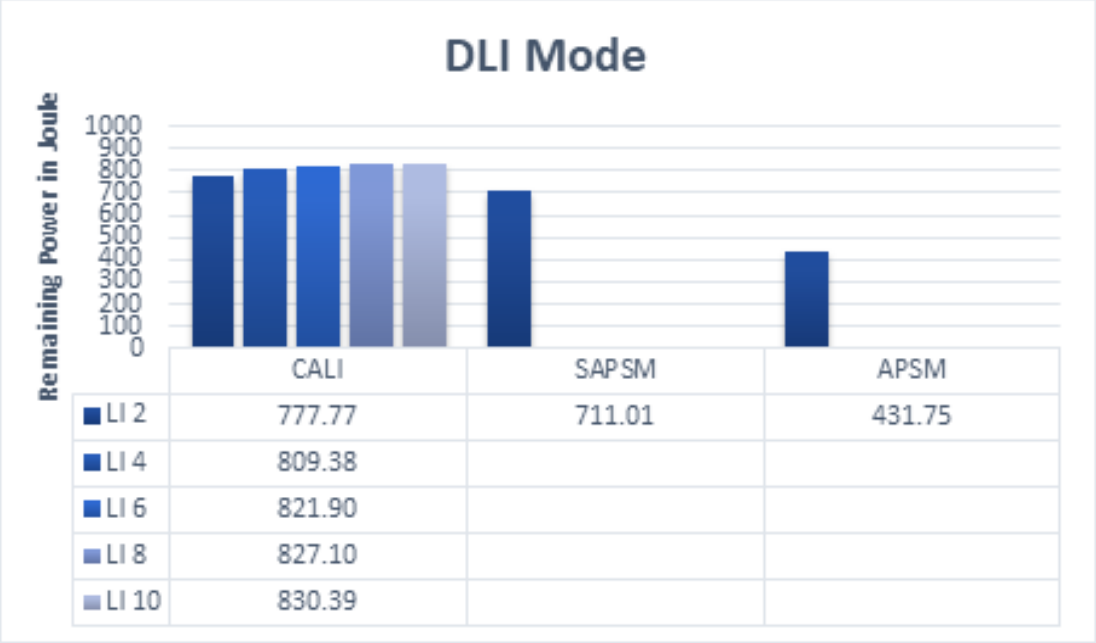


Figure 7. 5: Comparison of CALI, SAPSM, and APSM in DLI mode against set 2 of energy parameters

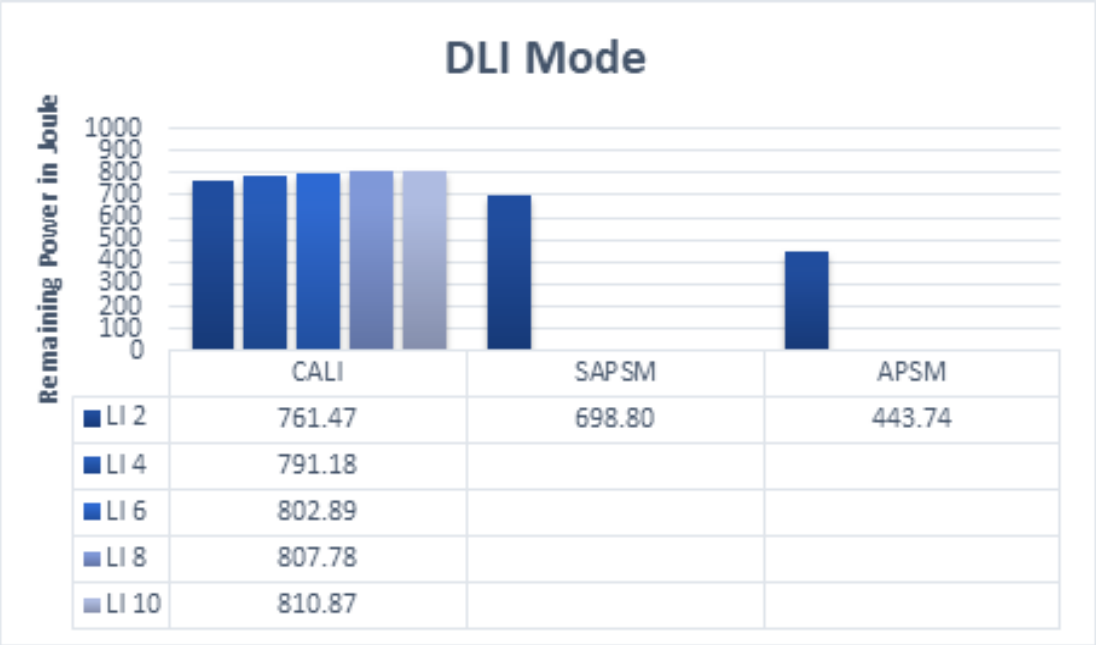


Figure 7. 6: Comparison of CALI, SAPSM, and APSM in DLI mode against set 3 of energy parameters

The levels of energy consumption of CALI, SAPSM and APSM in low mode against the 3 sets of energy parameters are shown in Figures 7.7, 7.8 and 7.9. For all 3 sets of energy parameters CALI consumes less energy than SAPSM and APSM. Low mode reflects applications with the lowest degree of interaction in the background. Where interactions mostly occur during fetching advertisements. In the experiments the listen interval of CALI was set to 20. Besides, after the playing time of 110 seconds when the network traffic to load the advertisements occurs, we also observed a small level of network interaction during playing time. While small this was sufficient to switch APSM to awake mode. In Set 2, CALI consumes 14.39% less energy compared to SAPSM and 41.83% when compared to APSM.

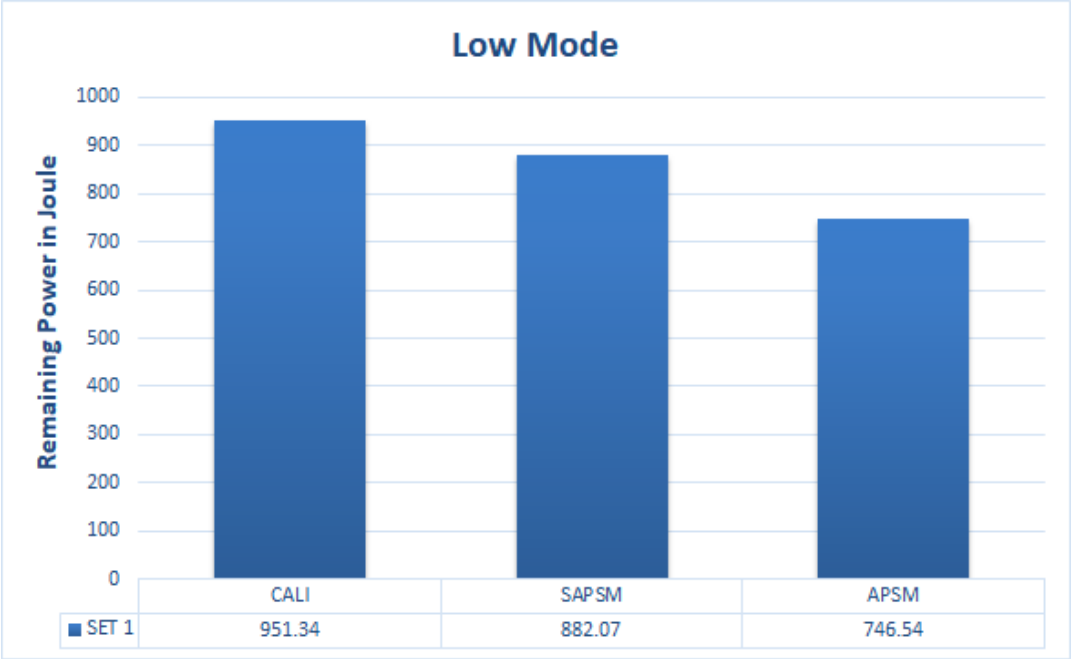


Figure 7. 7: Comparison of CALI, SAPSM, and APSM in low mode against set 1 of energy parameters

Figure 7. 8: Comparison of CALI, SAPSM, and APSM in low mode against set 2 of energy parameters

Figure 7. 9: Comparison of CALI, SAPSM, and APSM in low mode against set 3 of energy parameters

Figure 7.10 shows the levels of energy consumption of CALI, SAPSM, and APSM in awake mode for the 3 sets of energy parameters. As awake mode reflects applications with higher levels of network traffic, the WNIC is always on. Therefore, based on traffic of Skype video call, in all 3 sets of energy parameters, the levels of energy consumption of CALI, SAPSM, and APSM are identical. This is due to the behaviour of CALI, SAPSM and APSM to this type of traffic.

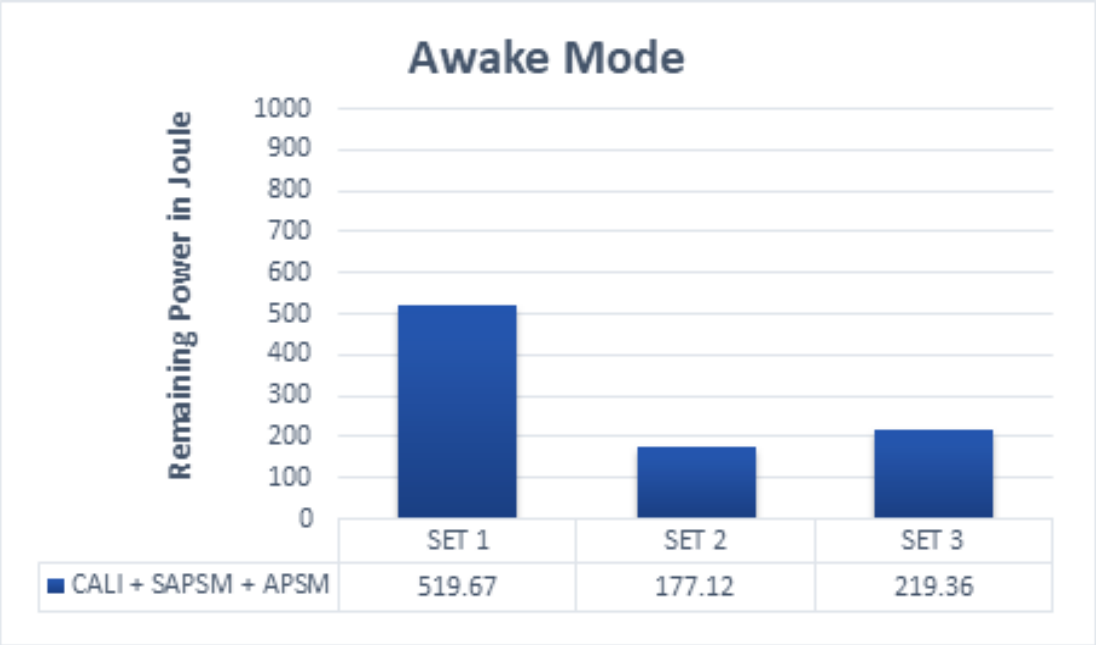


Figure 7. 10: Comparison of CALI, SAPSM, and APSM in awake mode against the 3 sets of energy parameters

7.2.3 Value Variations of Energy Parameters

Further investigation was carried out observing the behaviour of CALI, as we varied the values of individual energy parameters between their max and min across the three sets. We chose each individual energy parameter and gradually increased its value from the minimum as in Set 1 to match the max value as in Set 2. The values for the other energy parameters were kept unchanged.

Figures 7.11, 7.12, 7.13, 7.14 and 7.15 show the energy consumption of CALI in buffering mode as the value of the individual power parameters were varied.

Figure 7.11 shows the energy consumption of CALI in buffering mode for changing values of txPower 1.4W (Set 1), to 1.675W (Set 2). In this context, txPower reflects the energy consumption of the acknowledgment packets sent by the wireless device to an AP upon receiving the destined packets.

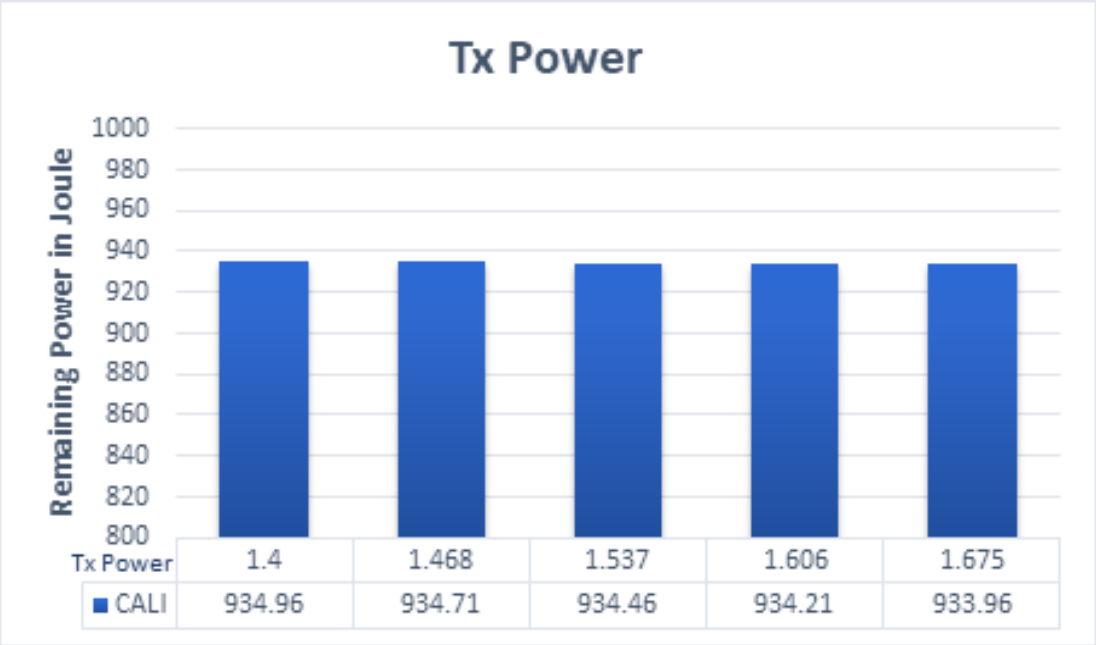


Figure 7. 11: Levels of energy consumption of CALI in buffering mode against the value variations of txPower energy parameter

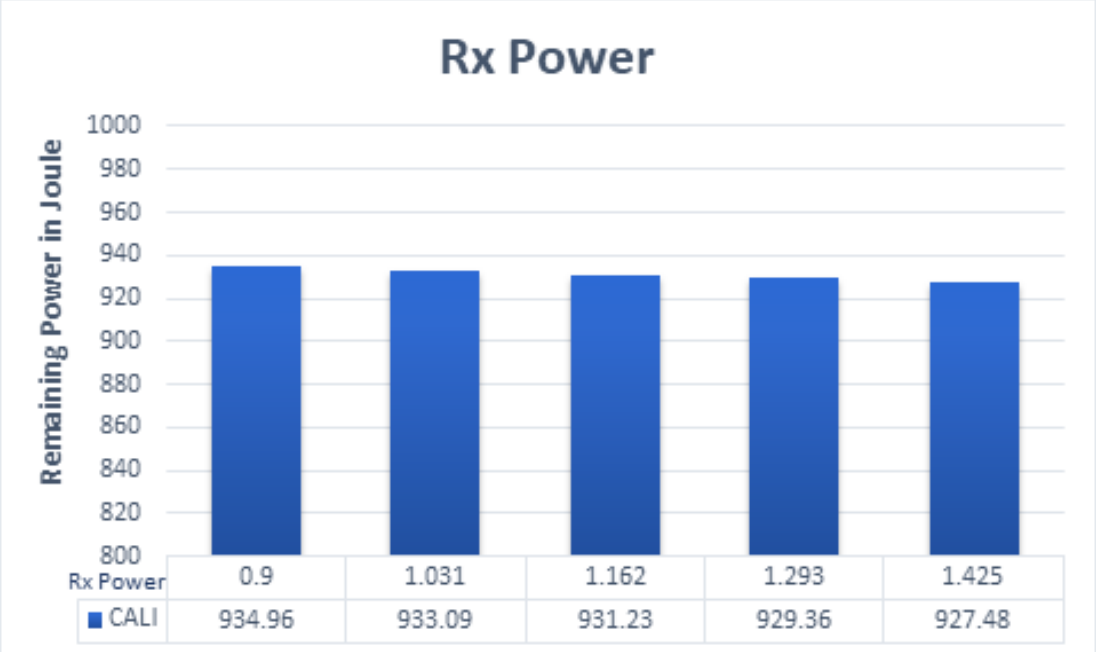


Figure 7. 12: Levels of energy consumption of CALI in buffering mode against the value variations of rxPower energy parameter

Figure 7. 13: Levels of energy consumption of CALI in buffering mode against the value variations of idlePower and transitionPower energy parameters

Figure 7.12 illustrates levels of energy consumption of CALI in buffering mode when incrementing rxPower from 0.9W (Set 1), to 1.425W (Set 2). rxPower reflects the energy consumption of the wireless device while receiving packets from an AP.

As mentioned before, the value of transitionPower must be twice that of idlePower. Therefore, we have incremented the values of transitionPower along with the value of idlePower.

Levels of energy consumption of CALI in buffering mode when incrementing transitionPower and idlePower from values in Set 1 to values in Set 2 are shown in Figure 7.13.

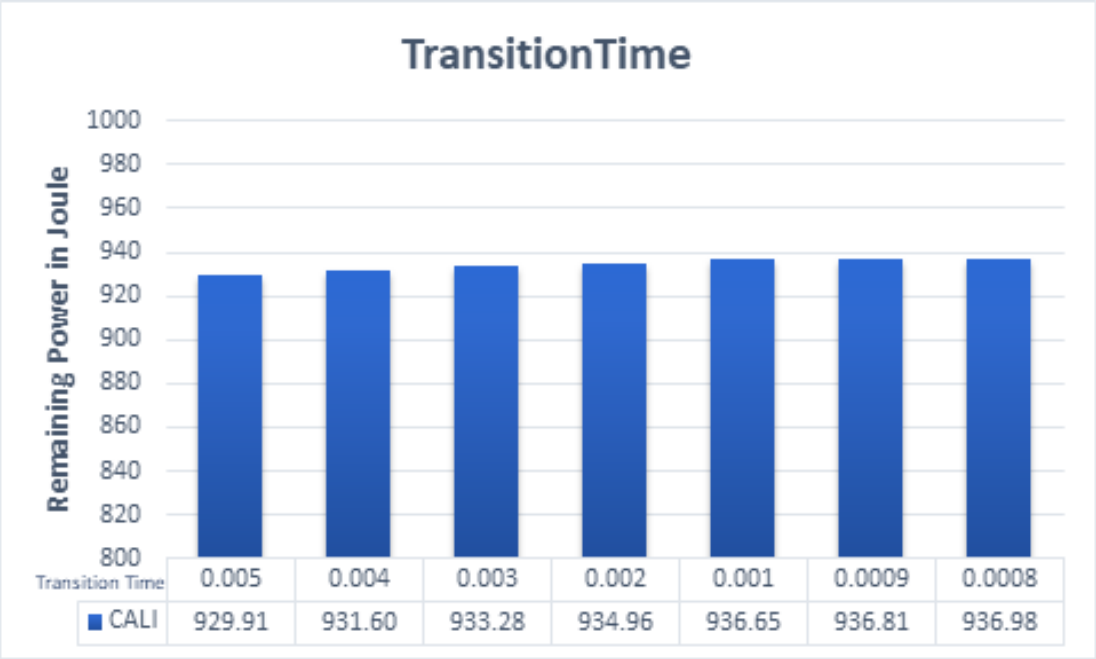


Figure 7. 14: Levels of energy consumption of CALI in buffering mode against the value variations of transitionTime

The transitionTime value is identical in all 3 sets of 0.002s. In order to further analyse its impact on energy consumption, we have varied transitionTime between 0.005s and 0.0008s. The impact of increasing and decreasing the transitionTime on energy consumption of CALI in buffering mode is shown in Figure 7.14.

Figure 7.15 shows levels of energy consumption of CALI in buffering mode while increasing sleepPower from 0.06W (Set 1), to 0.177W (Set 2). As can be expected, we observe that the value of the sleepPower parameter has a major impact on the levels of energy consumption of CALI in comparison to the other energy parameters.

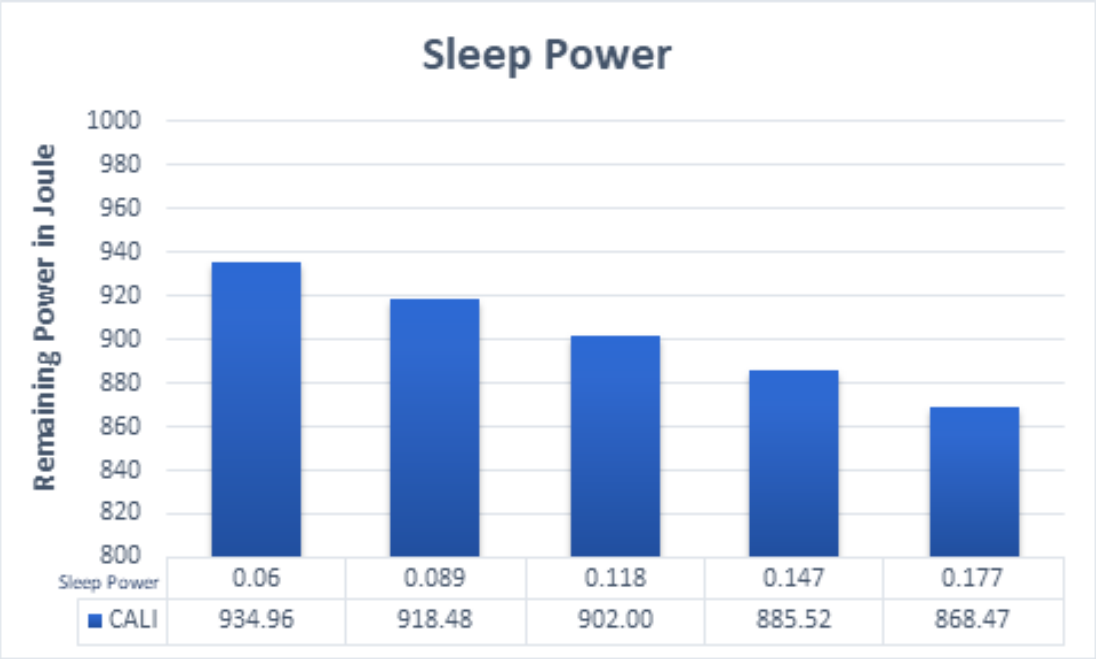


Figure 7. 15: Levels of energy consumption of CALI in buffering mode against the value variations of sleepPower energy parameter

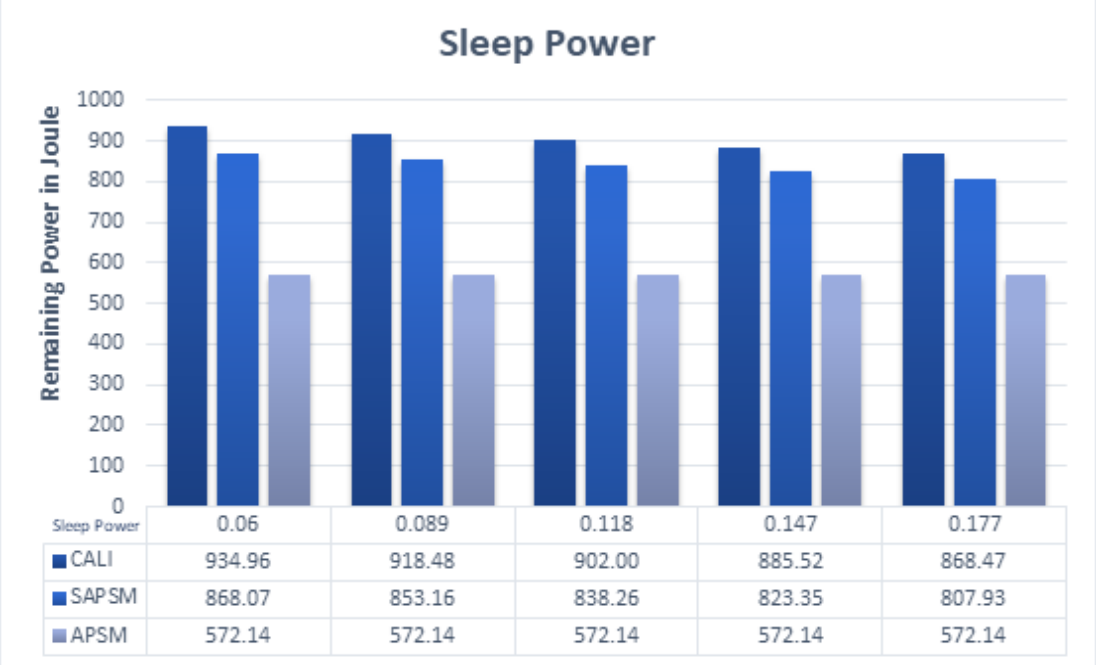


Figure 7. 16: Levels of energy consumption of CALI, SAPSM, and APSM in buffering mode against the value variations of sleepPower energy parameter

Figure 7.16 shows the levels of energy consumption of CALI, SAPSM, and APSM in buffering mode when increasing sleepPower from 0.06W (Set 1), to

0.177W (Set 2). CALI consumes less energy than SAPSM and APSM. The power consumption of APSM remains static, as the WNIC remains awake and thus the value of sleepPower has no impact on energy consumption.

7.3 Summary

In summary, this chapter evaluated the effect of adjusting the WNIC on energy consumption after the accomplishment of the classification process using an ML classification model. It also described the experimental setup used in the creation of the corresponding scenarios of CALI power saving modes. This was followed by assessing the performance of CALI power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, including APSM and SAPSM using the three sets of energy parameters. Furthermore, it assessed the performance of CALI against the value variations of energy parameters.

8

CONCLUSION AND FUTURE WORK

8.1 Introduction

This chapter concludes the thesis. It starts by summarising the motivation behind this research work and then discusses the main findings (section 8.2). section 8.3 revisits the research objectives and describes how they were fulfilled. Section 8.4 highlights the limitations and outlines possible future research directions.

8.2 Thesis Summary

Regardless of the rapid growth and popularity of WLANs, the energy consumption caused by WNIC during wireless communication remains a significant factor in reducing the battery life of power-constrained wireless devices.

The authors of [10] proposed SAPSM. SAPSM replaced the threshold mechanism of APSM with a set of two priorities, high and low. Thus, each network based smartphone application is labelled as high and low, with the aid of an ML classifier. Consequently, for applications set as a high priority, the WNIC switches into awake mode and remains in SPSM with low priority applications.

However, no additional priority or mode has been proposed, e.g., for applications with very low levels of network interactivity or applications using buffer streaming.

Comparing with SAPSM, this thesis has extended the number of categories by considering a varied range of smartphone applications' network traffic that reflect a diverse array of network behaviour and interactions.

Hence, the aim of this thesis was to develop a power saving mechanism that optimises the sleep and awake cycles of the WNIC in accordance with smartphone applications' network traffic, reflecting a diverse array of network behaviour and interactions.

In this thesis, we have developed a Context-Aware Listen Interval (CALI), in which the wireless network interface, with the aid of an ML classification model, sleeps and awakes based on the level of network activity of each application.

Firstly, we introduced a context-aware network traffic classification approach based on ML classifiers to classify the network traffic of wireless devices in WLANs. Different levels of traffic behaviour and interaction were contextually exploited for the classification by the application of ML classifiers.

A real-world dataset is recorded, based on nine smartphone applications' network traffic, reflecting different types of network behaviour and interaction. This is used firstly to evaluate the performance of five ML classifiers using 10-fold cross-validation, this was followed by conducting extensive experimentation to determine whether the selected classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data.

In the first experiment, a representative application from each class was selected for training the ML classifiers, and their generalisation capacity was evaluated on different applications that were not included in the training data. In terms of generalising to unseen testing data of class high, the results of the first experiment showed that the learned classification models were only

capable of capturing the variance in the traffic range of video call applications and that by generalising well to unseen testing data of Google Hangouts video call only. But, in terms of the voice call applications, the resultant classification models were incapable to generalise to testing data of both Skype and Google Hangouts voice calls.

The second experiment was performed to assess the generalisation capacity of the learned classification models on extended training data, particularly after the inclusion of the Skype voice call application into the training data. Where the experimental results showed an improvement in the generalisation performance specifically on testing data of class high, this was due to the training of the ML classifiers on a wider variation range that resulted from combining the training data of Skype video call and Skype voice call.

The third experiment was performed to assess the generalisation capacity of the learned classification models on reduced training data. Where the training samples of each application used for training the ML classifiers in the second experiment were reduced to half and then to a quarter. However, the results of this experiment showed that reducing the amount of training data has a minimal impact on the generalisation performance, but still, better generalisation performance can be achieved by training with more samples.

To further assess the generalisation capacity, the fourth experiment was performed by switching the training and testing data that were used in the previous experiments 2 and 3. This involved training the ML classifiers on applications that were previously used for testing and assessing their generalisation performance on applications that were used for training. However, the results of this experiment showed that training the ML classifiers on training data with a wider variation in the traffic range leads to better generalisation performance.

Moreover, given that the classification models in the second, third and fourth experiments were capable of achieving high results on unseen testing data of applications that were not included in the training data. This thesis further explored the feasibility of manually crafting rules to hand-classify the training data. Where attempt was made to hand-classify the training data and the outcomes were subsequently discussed and compared with the benefits offered by classification models constructed using ML classifiers.

In addition, this thesis conducted the hyperparameter optimisation process using both manual and automated tuning methods to identify the optimal settings that result in a better-performing classification model. Where various hyperparameter settings were explored by performing 10-fold cross-validation firstly on the training data of experiment three consisting of 185 samples. Followed by evaluating the performance of the constructed classification models using the obtained optimal sets of hyperparameter values on the testing data of the same experiment.

This thesis further assessed the performance of the classification models by repeating the previous four experiments using the optimal sets of hyperparameter values that were obtained through the optimisation process. Where the experimental results particularly of the repeated experiments one and four showed that using the optimised hyperparameters for a particular training data may not always lead to improved model performance when there are changes in the overall distribution of new training data, and the default hyperparameter settings in some cases perform comparably or better than the optimised hyperparameters.

Thus, further hyperparameter tuning was performed in this thesis, where the optimal sets of hyperparameter values were determined for classification models of the first and fourth experiments and the experimental results

confirmed that better results can be obtained by conducting a hyperparameter optimisation process independently for each training data.

Moreover, in order to optimise the sleep and awake cycles of the WNIC in accordance with the smartphone applications' network activity, we have developed four CALI power saving modes.

These power saving modes enabled additional power saving opportunities and have been devised based on the classified output traffic of the captured samples from the nine smartphone applications' network traffic. Hence, the ML classification model classifies the new unseen samples into one of the modes, where the WNIC will be adjusted to operate into one of CALI power saving modes.

Moreover, CALI handles applications, which it cannot map to one of the four modes by reverting the WNIC to operate in SPSM mode. That means, the worst possible performance is that of SPSM, but if one of the four modes applies, a significant performance improvement with respect to power saving is achieved.

We evaluated the performance of CALI power saving modes, by comparing the levels of energy consumption with existing benchmark power saving approaches, including APSM and SAPSM using varied sets of energy parameters. And the experimental results have demonstrated that CALI consumes up to 75% less power when compared to APSM, and up to 14% less energy when compared to SAPSM power saving approach.

Lastly, our approach relies on an ML classification model to optimise the energy efficiency of power-constrained wireless devices. Therefore, the computational cost of training and testing the ML classifier is crucial. In this research, we have demonstrated high accuracy and low computational cost for building a classification model. Clearly, this is a one-off cost during

deployment. In addition, the cost of our approach at runtime is minimal as the WNIC simply operates in one of the CALI power saving modes, once the classification of the traffic is completed.

8.3 Meeting the Objectives

This section discusses how the research objectives stated in chapter 1 have been satisfied and addressed during our research.

- **Identify and construct a real-world dataset based on a varied range of smartphone applications' network traffic depicting different types of network behaviour and interaction.**

  Section 4.4 describes the process of data extraction and preparation employed in this research for constructing the dataset. The dataset was constructed by capturing real-time instances of network traffic from nine selected smartphone applications depicting varied types of network behaviour and interaction; including, two VoIP applications, two applications of video calls, two applications of intermittent network interaction, two applications of very low network interaction, and finally one application representing applications with buffer streaming capabilities. This has resulted in the construction of a dataset, named Dataset 1, consisting of 1350 instances, with 150 instances per application and 6 features per instance. Section 4.3 begins by justifying the selection of the chosen applications and the assignment of output classes. Where four output classes were assigned to cater for the network traffic of these applications. Thereby out of the nine chosen applications, the first output class was assigned to the four applications that represent real-time applications with high and constant levels of network interaction. The reason for having four applications for this output class is to ensure more variation in the range of network traffic included in the training data by

having two VoIP applications and two video-calling applications. For the remaining three types of network traffic, the second output class was assigned to the two applications that represent network traffic with intermittent levels of interaction, while the third output class was assigned to the two applications that represent the least levels of network interaction. Finally, the fourth output class was assigned to one application that represents the network traffic of audio streaming applications. Section 4.5.1 describes the construction of further datasets from Dataset 1 by the application of different feature selection algorithms, Dataset 2CBFS is based on a consistency feature selection algorithm and Dataset 3IGFS is based on an information gain feature selection algorithm.

- **Train ML classifiers to learn mapping the input features of each sample to an output class from the training set and build an ML classification model.**

Section 4.2 discuss how the network traffic of the nine smartphone applications reflecting a diverse array of network behaviour and interaction were exploited to provide the contextual inputs for training ML classifiers of the output traffic, thus building an ML classification model which is capable of classifying the new unseen samples into one of the classes. The set of six input features are: 1- receiving data rate in Kbytes/sec. 2- transmitting data rate in Kbytes/sec. 3- total received Kbytes. 4- total transmitted Kbytes. 5- total number of received packets. 6- total number of transmitted packets. These features were used as contextual inputs for training ML classifiers of output classes: 1- high. 2- varied. 3- low. 4- buffering.

- **Evaluate the performance of ML classifiers using 10-fold cross-validation. Based on the result of the analysis, determine the more suitable ML classifier for classifying smartphone applications' network traffic reflecting varied types of network behaviour and interaction. Then, assess the generalisation capacity of the selected classification models on unseen testing data of applications that were not included in training data. Along with evaluation metrics, provide a confusion matrix to enable a detailed breakdown of the predictions, including the distribution of correct and incorrect predictions made by the classification models.**

Section 4.5.2 evaluates the performance of the five commonly used ML classifiers described in section 3.6. The applied ML classifiers were MLP, KNN, SVM, decision tree (C4.5), and Random forest. And the performance of each classifier was evaluated on Dataset 1, Dataset 2CBFS and Dataset 3IGFS using 10-fold cross validation, in terms of classification accuracy, precision, recall and f-measure. This section also assessed the processing time to build a classification model. Comparing the results obtained for the five ML classifiers in all datasets in terms of all evaluation metrics, we found that a number of effective features can be considered to improve the overall results. Moreover, we conclude that the optimum results in terms of all evaluation metrics used in these experiments were achieved by KNN in Dataset 3IGFS using10-fold cross-validation. We determined KNN to be the best ML classifier in terms of classifying smartphone applications' network traffic based on different levels of behaviour and interaction. This was followed by conducting extensive experimentation in sections 5.2 to 5.5 to determine whether the selected classification models not only perform well on training data but also generalise well on unseen testing data of applications that were not included in training data. Where the performance of each classifier is evaluated in terms of classification accuracy, macro-average of precision, recall and weighted average f-

measure. Along with evaluation metrics, a confusion matrix was provided to enable a detailed breakdown of the predictions, including the distribution of correct and incorrect predictions made by the classification models. In the first experiment, a representative application from each class was selected for training the ML classifiers, and their generalisation capacity was evaluated on different applications that were not included in the training data. In terms of generalising to unseen testing data of class high, the results of the first experiment showed that the learned classification models were only capable of capturing the variance in the traffic range of video call applications and that by generalising well to unseen testing data of Google Hangouts video call only. But, in terms of the voice call applications, the resultant classification models were incapable to generalise to testing data of both Skype and Google Hangouts voice calls. The second experiment was performed to assess the generalisation capacity of the learned classification models on extended training data, particularly after the inclusion of the Skype voice call application into the training data. Where the experimental results showed an improvement in the generalisation performance specifically on testing data of class high, this was due to the training of the ML classifiers on a wider variation range that resulted from combining the training data of Skype video call and Skype voice call. The third experiment was performed to assess the generalisation capacity of the learned classification models on reduced training data. Where the training samples of each application used for training the ML classifiers in the second experiment were reduced to half and then to a quarter. However, the results of this experiment showed that reducing the amount of training data has a minimal impact on the generalisation performance, but still, better generalisation performance can be achieved by training with more samples. To further assess the generalisation capacity, the fourth experiment was performed by switching the training and testing data that were used in the previous

experiments 2 and 3. This involved training the ML classifiers on applications that were previously used for testing and assessing their generalisation performance on applications that were used for training. However, the results of this experiment showed that training the ML classifiers on training data with a wider variation in the traffic range leads to better generalisation performance. Moreover, given that the classification models in the second, third and fourth experiments were capable of achieving high results on unseen testing data of applications that were not included in the training data. Section 5.6 further explored the feasibility of manually crafting rules to hand-classify the training data. Where attempt was made to hand-classify the training data and the outcomes were subsequently discussed and compared with the benefits offered by classification models constructed using ML classifiers.

- **Devise power saving modes based on the classified output traffic of the captured samples from a varied range of smartphone applications' network traffic.**

Section 4.3 discusses how the CALI power saving modes were employed for optimising the sleep and awake cycles of the WNIC in accordance with the smartphone applications' network activity. We introduced four CALI power saving modes based on the classified output traffic of the captured samples from the nine smartphone applications' network traffic. Therefore, once the ML classification model classifies the new unseen samples into one of the classes, the WNIC is adjusted to operate into one of CALI power saving modes. In addition, CALI handles smartphone applications, which it cannot map to one of the four modes by reverting the WNIC to operate in SPSM mode. That means, the worst possible performance is that of SPSM, but if one of the four modes applies, a significant performance improvement with respect to power saving is achieved.

- **Evaluate the performance of the proposed power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, using varied sets of energy parameters.**

Subsection 7.2.2 evaluates the performance of CALI power saving modes by comparing the levels of energy consumption with existing benchmark power saving approaches, using varied sets of energy parameters. We selected APSM as the most current power saving approach deployed in smartphones and SAPSM as a recent technique also employing ML. The experimental results show that CALI consumes up to 75% less power when compared to APSM, and up to 14% less energy when compared to SAPSM power saving approach. This is followed by assessing the performance of CALI against the value variations of energy parameters in subsection 7.2.3.

8.4 Limitations and Future Work

This section highlights the limitations and outlines possible future research directions.

8.4.1 Limitations

- **Dataset**

In this research, we investigated the network activity of a single smartphone application opened at a given time without considering the network activity of applications that run simultaneously. Where the classification models were constructed from the training samples captured from the network traffic of nine smartphone applications that were running individually.

Therefore, there is room for further investigation that can be carried out by capturing the network traffic of applications with different behaviours and

interactions. For example, to represent the network traffic of applications that run simultaneously, additional samples can be recorded from both an audio streaming app running in the background, along with a lower network usage gaming app and then added to the dataset.

Moreover, to represent the network traffic of audio streaming applications with playback buffering capacity, in this research, we chose a radio station streaming at 128 kbps. However, additional samples from a radio station streaming at 320 kbps or higher could be recorded and then added to the dataset.

- **Power saving modes**

There is a scope for further investigation to find the optimal number of power saving modes that could be associated with the CALI's framework. This would be based on further analysis of the captured network traffic of applications with different behaviours and interactions.

For example, based on the analysed network traffic of applications that run simultaneously, an additional power saving mode can be introduced with its listening interval resides between the current low and buffering modes. Similarly, based on the analysed network traffic of radio station streaming at 320 kbps or higher, an additional CALI power saving mode can be developed and incorporated into CALI's framework.

- **The framework**

The ML based classification models employed in this research were capable of achieving high results on unseen testing data of applications that were not included in the training data. Therefore, we explored the feasibility of manually crafting rules to hand-classify the training data. Where attempt was made to hand-classify the training data and the

outcomes were subsequently discussed and compared with the benefits offered by classification models constructed using ML classifiers. However, it also can be worthwhile to employ simpler mechanisms such as thresholding as an approach to classify network traffic of wireless devices in WLANs.

Moreover, the application of the CALI power saving framework is limited to wireless devices operating on WLANs and this possibly could be extended to other similar types of wireless networks such as 4G/5G mobile data networks and WiMAX.

Furthermore, a context-aware power saving framework could also potentially be applied to other devices like Internet of Things (IoT) appliances; as long as these devices have or operate on a set of finite modes or states, and there are inputs that cause the transition into one of states or modes. However, it is challenging but worth trying to design a context-aware power saving framework for devices or appliances that do not have a finite set of states and are always on e.g., battery chargers.

- **WEKA tool**

Weka tool may not provide customisation options for certain parameters, e.g., the only activation function for an MLP is sigmoid. However, the tool was sufficient for training and hyperparameter optimisation, and this did not restrict from obtaining optimum results.

8.4.2 Future Work

This section outlines possible future research directions based on the above limitations.

- **Larger dataset**

Future work should consider investigating, training, and evaluating ML classifiers on a larger dataset containing samples of network traffic captured from a wider range of smartphone applications , these include samples captured from Video on Demand (VoD) applications streaming 4K/8k Ultra HD video content, samples captured from applications running simultaneously, and samples captured from audio streaming applications e.g., radio stations streaming at 320 kbps or higher.

- **Additional power saving modes**

Future work should also investigate the possibility of developing additional CALI power saving modes based on the classified output traffic from further smartphone applications' network traffic.

- **Energy optimisation**

In this research we developed a context-aware listen interval to optimise energy efficiency of power-constrained wireless devices in WLANs. However, future work should include implementation and evaluation of the CALI power saving framework on wireless networks similar to WLANs such as 4G/5G mobile data networks and WiMAX. Also, it will be worthwhile investigating a context-aware power saving framework for continuously variable devices.

- **Real implementation**

Another possible future study would be to implement the CALI power saving approach in a real environment. Although real deployment is complex, costly, and time-consuming. However, it would provide a better insight and more realistic results.

REFERENCES

[1] F. Wu et al., “Named Data Networking Enabled Power Saving Mode Design for WLAN,” IEEE Transactions on Vehicular Technology, vol. 69, no. 1, pp. 901-913, 2020.

[2] “IEEE 802.11-2012: Wireless LAN Medium Access Control MAC and Physical Layer PHY Specifications,” IEEE 802.11 LAN Standards 2012, 2012.

[3] S. Baek and B. D. Choi, "Performance analysis of power save mode in IEEE 802.11 infrastructure WLAN", Int. Conf. Telecommun., pp. 1-4, Jun. 2008.

[4] N. Ding, A. Pathak, D. Koutsonikolas, C. Shepard, Y. C. Hu and L. Zhong, “Realizing the full potential of PSM using proxying,” in 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 2012, pp. 2821-2825.

[5] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, “NAPman: Network assisted power management for WiFi devices,” in Proc. 8th Int. Conf. Mobile Syst., Appl., Services, pp. 91-106, Jun. 2010.

[6] A. Pyles, D. T. Nguyen, X. Qi and G. Zhou, “Bluesaver: A Multi-PHY Approach to Smartphone Energy Savings,” IEEE Transactions on Wireless Communications, vol. 14, no. 6, pp. 3367-3377, Jun. 2015.

[7] V. Bernardo, B. Correia, M. Curado, and T. I. Braun, “Towards end-user driven power saving control in Android devices,” in Internet of Things, Smart Spaces, and Next Generation Networks and Systems (Lecture Notes in Computer Science), vol. 8638. Cham, Switzerland: Springer, 2014, pp. 231–244. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-10353-2_20#aboutcontent, doi: 10.1007/978-3-319-10353-2_20.

[8] A. J. Pyles, Z. Ren, G. Zhou and X. Liu, "SIFI: Exploiting voip silence for WiFi energy savings insmart phones", Proc. 13th Int. Conf. Ubiquitous Comput., pp. 325-334, 2011.

[9] Kwon, H., Kim, S., Son, Y., Yang, C., Byeon, S. and Choi, S., 2019, November. AWARE: Adaptive Wi-Fi Power Save Operation Coexisting with LTE-U. In 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS) (pp. 208-216). IEEE.

[10] A. J. Pyles, X. Qi, G. Zhou, M. Keally and X. Liu, "SAPSM: Smart adaptive 802.11 PSM for smartphones," in Proc. ACM Conf. Ubiquitous Comput., pp. 11-20, Sept. 2012.

[11] X. Xia, S. Li, Y. Zhang, B. Li, Y. Zheng and T. Gu, "Enabling Out-of-Band Coordination of Wi-Fi Communications on Smartphones," IEEE/ACM Transactions on Networking, vol. 27, no. 2, pp. 518-531, Apr. 2019.

[12] Pérez-Costa, X. and Camps-Mur, D., 2006, June. AU-APSD: Adaptive IEEE 802.11 e unscheduled automatic power save delivery. In 2006 IEEE International Conference on Communications (Vol. 5, pp. 2020-2027). IEEE.

[13] Westcott, D.A., Coleman, D.D., Miller, B. and Mackenzie, P., 2011. CWAP Certified Wireless Analysis Professional Official Study Guide: Exam PW0-270. John Wiley & Sons.

[14] Pérez-Costa, X. and Camps-Mur, D., 2010. IEEE 802.11 E QoS and power saving features overview and analysis of combined performance [Accepted from Open Call]. Wireless Communications, IEEE, 17(4), pp.88-96.

[15] R. Prasad, A. Kumar, R. Bhatia, "Electronics, Computing and Communication Technologies (IEEE CONECCT), 2014 IEEE International Conference on," Ubersleep: An innovative mechanism to save energy in IEEE 802.11 based WLANs, Vol., no. , pp. 1 - 6, 6-7 Jan. 2014.

[16] IEEE Computer Society LAN MAN Standards Committee, Amendment to IEEE Standard 802.11: Sub 1 GHz License Exempt Operation, IEEE Standard 802.11ah, Feb. 2016

[17] H. Yang, D. Deng and K. Chen, "On Energy Saving in IEEE 802.11ax," in IEEE Access, vol. 6, pp. 47546-47556, 2018, doi: 10.1109/ACCESS.2018.2865763.

[18] Y. Li, X. Zhang and K. L. Yeung, "DLI: A dynamic listen interval scheme for infrastructure-based IEEE 802.11 WLANs," IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Hong Kong, China, 2015, pp. 1206-1210.

[19] Y. Li, G. Zhou, G. Ruddy, and B. Cutler, "A Measurement-Based Prioritization Scheme for Smartphone Applications," Wireless personal communications, vol. 78, no. 1, pp.333-346, Sept. 2014.

[20] Kwon, S.W. and Cho, D.H., 2006, September. Efficient power management scheme considering inter-user QoS in wireless LAN. In Vehicular Technology Conference, 2006. VTC-2006 Fall. 2006 IEEE 64th (pp. 1-5). IEEE.

[21] Fahad R. Dogar, Peter Steenkiste, and Konstantina Papagiannaki. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys 2010.

[22] K. J. Jang, S. Hao, A. Sheth, and R. Govindan "Snooze: Energy management in 802.11n WLANs," in Proc. CoNEXT, Dec. 2011, Art. no. 12.

[23] A. J. Pyles, Z. Ren, G. Zhou, and X. Liu, "SiFi: exploiting VoIP silence for WiFi energy savings in smart phones," in Proc. UbiComp, Sep. 2011, pp. 325–334.

[24] J. Liu and L. Zhong, "Micro power management of active 802.11 interfaces," in Proc. MobiSys, Jun. 2008, pp. 146–159.

[25] Y. Xie, X. Luo and R. K. C. Chang, "Centralized PSM: An AP-centric power saving Mode for 802.11 infrastructure networks," 2009 IEEE Sarnoff Symposium, 2009, pp. 1-5, doi: 10.1109/SARNOF.2009.4850348.

[26] E. Tan, L. Guo, S. Chen and X. Zhang, "PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs," 2007 IEEE International Conference on Network Protocols, 2007, pp. 123-132, doi: 10.1109/ICNP.2007.4375843.

[27] Y. He and R. Yuan, "A Novel Scheduled Power Saving Mechanism for 802.11 Wireless LANs," in IEEE Transactions on Mobile Computing, vol. 8, no. 10, pp. 1368-1383, Oct. 2009, doi: 10.1109/TMC.2009.53.

[28] P. Agrawal, A. Kumar, J. Kuri, M. K. Panda, V. Navda and R. Ramjee, "OPSM - Opportunistic Power Save Mode for Infrastructure IEEE 802.11 WLAN," 2010 IEEE International Conference on Communications Workshops, 2010, pp. 1-6, doi: 10.1109/ICCW.2010.5503903.

[29] K. Omori, Y. Tanigawa, H. Tode, "Power-Saving for Wireless Stations Using RTS/CTS Handshake and Burst Transmission in Wireless LANs," 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, pp. 708-711, Jul. 2017.

[30] K. Omori, Y. Tanigawa and H. Tode, "A study on power saving using RTS/CTS handshake and burst transmission in wireless LAN," 2015 10th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT), Colombo, pp. 1-3, Aug. 2015.

[31] Justin Manweiler and Romit Roy Choudhury. Avoiding the rush hours: Wifi energy management via traffic isolation. In Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys 2011.

[32] D. Liu, H. Wang, G. Zhou, W. Mao, and B. Li, "Arbitrating traffic contention for power saving with multiple PSM clients," IEEE Trans. Wireless Commun., vol. 15, no. 10, pp. 7030–7043, Oct. 2016.

[33] G. M. Fisher, "Remembering mollie orshansky—The developer of the poverty thresholds," Soc. Secur. Bull., vol. 68, no. 3, pp. 78–83, 2008.

[34] Z. Zeng, Y. Gao and P. R. Kumar, "SOFA: A Sleep-Optimal Fair-Attention Scheduler for the Power-Saving Mode of WLANs," 2011 31st International Conference on Distributed Computing Systems, 2011, pp. 87-98, doi: 10.1109/ICDCS.2011.78.

[35] Hsiao-Po Lin, Shih-Chang Huang, and Rong-Hong Jan. A power-saving scheduling for infrastructure-mode 802.11 wireless LANs. Comput. Commun., 29(17):3483–3492, November 2006.

[36] H. Han, Y. Liu, G. Shen, Y. Zhang, Q. Li and C. C. Tan, "Design, Realization, and Evaluation of DozyAP for Power-Efficient Wi-Fi Tethering," in IEEE/ACM Transactions on Networking, vol. 22, no. 5, pp. 1672-1685, Oct. 2014, doi: 10.1109/TNET.2013.2283636.

[37] Y. Zhang and Q. Li, "Exploiting ZigBee in Reducing WiFi Power Consumption for Mobile Devices," in IEEE Transactions on Mobile Computing, vol. 13, no. 12, pp. 2806-2819, 1 Dec. 2014, doi: 10.1109/TMC.2014.2315788.

[38] G. Ananthanarayanan and I. Stoica, “Blue-Fi: Enhancing Wi-Fi performance using Bluetooth signals,” in Proc. ACM MobiSys, 2009, pp. 249–262.

[39] E.J. Vergara and S. Nadjm-Tehrani. Watts2share: Energy-aware traffic consolidation. In Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, pages 14–22, Aug 2013.

[40] J. Chung, J. Park, C. Kim and J. Choi, "C-SCAN: Wi-Fi Scan Offloading via Collocated Low-Power Radios," in IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1142-1155, April 2018, doi: 10.1109/JIOT.2018.2811240.

[41] F. Lu, G. M. Voelker, and A. C. Snoeren, “SloMo: Downclocking WiFi communication,” in Proc. USENIX NSDI, pp. 255–258, 2013.

[42] X. Zhang and K. G. Shin, “E-MiLi: Energy-Minimizing Idle Listening in WirelessNetworks,” IEEE Transactions on Mobile Computing, vol. 11, no. 9, pp. 1441-1454, Sept. 2012.

[43] W. Wang, Y. Chen, L. Wang and Q. Zhang, “Sampleless Wi-Fi: Bringing Low Power to Wi-Fi Communications,” IEEE/ACM Transactions on Networking, vol. 25, no. 3, pp. 1663-1672, Jun. 2017.

[44] C. Deng et al., “IEEE 802.11be Wi-Fi 7: New Challenges and Opportunities,” IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2136-2166, 2020.

[45] Wi-fi.org. 2021. certification url check | Wi-Fi Alliance. [online] Available at: https://www.wifi.org/download.php?file=/sites/default/files/private/Economic_Value_of_Wi-Fi_Highlights_202109_0.pdf. [Accessed 13 September 2021].

[46] J. Saldana et al., “Attention to Wi-Fi Diversity: Resource Management in WLANs With Heterogeneous APs,” IEEE Access, vol. 9, pp. 6961-6980, Jan. 2021.

[47] X. Xie, W. Yang and K. Tian, “Delay-aware Power Saving Mechanism for 802.11 Wireless LANs via NDN,” 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), Shenzhen, 2018, pp. 154-159.

[48] M. F. Tuysuz, “Towards providing optimal energy-efficiency and throughput for IEEE 802.11 WLANs,” International Journal of Communication Systems, vol. 31, no. 13, pp. e3725, Jun. 2018.

[49] M. Jeruchim, P. Balaban, and K. Shanmugan, ‘‘Simulation of communication systems: Modeling, methodology and techniques,’’ in Information Technology: Transmission, Processing and Storage. New York, NY, USA: Springer, 2006.

[50] A. R. Khan, S. M. Bilal and M. Othman, "A performance comparison of open source network simulators for wireless networks," 2012 IEEE International Conference on Control System, Computing and Engineering, 2012, pp. 34-38, doi: 10.1109/ICCSCE.2012.6487111.

[51] “The Network Simulator-ns-2.” [Online]. Available: https://www.isi.edu/nsnam/ns/.

[52] “OMNET++ network simulator.” [Online]. Available: https://omnetpp.org/.

[53] “Riverbed (OPNET).” [Online]. Available: https://www.riverbed.com/products/steelcentral/opnet.html?redirect=opnet

[54] “QualNet.” [Online]. Available: https://www.scalable-networks.com/products/qualnet-network-simulation-software/.

[55] M. Fujinami, Y. Miyamoto and T. Murakami, “Wireless LAN Power Management Extension for ns-2”[Online]. Available: http://nspme.sourceforge.net/.

[56] A. Ksentini and Y. Hadjadj-Aoul, “QoE-based energy conservation for VoIP over WLAN,” 2012 IEEE Wireless Communications and Networking Conference (WCNC), Paris, France, 2012, pp. 1692-1697.

[57] A. Saeed and M. Kolberg, "Towards optimizing WLANs power saving: Novel context-aware network traffic classification based on a machine learning approach", IEEE Access, vol. 7, pp. 3122-3135, 2019.

[58] Al-Obeidat, F., El-Alfy, ES.M. Hybrid multicriteria fuzzy classification of network traffic patterns, anomalies, and protocols. Pers Ubiquit Comput 23, 777–791 (2019). https://doi.org/10.1007/s00779-017-1096-z.

[59] A. Dainotti, A. Pescape and K. C. Claffy, "Issues and future directions in traffic classification," in IEEE Network, vol. 26, no. 1, pp. 35-40, January-February 2012, doi: 10.1109/MNET.2012.6135854.

[60] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin and J. Aguilar, "Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey," in IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1988-2014, Secondquarter 2019, doi: 10.1109/COMST.2018.2883147.

[61] Tahaei, H., Afifi, F., Asemi, A., Zaki, F., & Anuar, N. B. (2020). The rise of traffic classification in IoT networks: A survey. Journal of Network and Computer Applications, 154, 102538.

[62] Z. Md. Fadlullah, F. Tang, B. Mao, J. Liu and N. Kato, "On Intelligent Traffic Control for Large-Scale Heterogeneous Networks: A Value Matrix-Based Deep Learning Approach," in IEEE Communications Letters, vol. 22, no. 12, pp. 2479-2482, Dec. 2018, doi: 10.1109/LCOMM.2018.2875431.

[63] Al Khater, N. and Overill, R.E., 2015, October. Network traffic classification techniques and challenges. In 2015 Tenth international conference on digital information management (ICDIM) (pp. 43-48). IEEE.

[64] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," in IEEE Communications Surveys & Tutorials, vol. 10, no. 4, pp. 56-76, Fourth Quarter 2008, doi: 10.1109/SURV.2008.080406.

[65] Shafiq, M., Tian, Z., Bashir, A. K., Jolfaei, A., & Yu, X. (2020). Data mining and machine learning methods for sustainable smart cities traffic classification: A survey. Sustainable Cities and Society, 60, 102177.

[66] Moore A, Papagiannaki K. “Toward the accurate identification of network applications”, Proceedings of Passive and Active Measurement Workshop (PAM2005). Boston(USA), 2005.

[67] A. Madhukar and C. Williamson, "A Longitudinal Study of P2P Traffic Classification," 14th IEEE International Symposium on Modeling, Analysis, and Simulation, 2006, pp. 179-188, doi: 10.1109/MASCOTS.2006.6.

[68] T. Bujlow, V. Carela-Español and P. Barlet-Ros, "Independent comparison of popular DPI tools for traffic classification", Comput. Netw., vol. 76, pp. 75-89, Jan. 2015.

[69] Pacheco, F., Exposito, E. and Gineste, M., 2020. A framework to classify heterogeneous Internet traffic with Machine Learning and Deep Learning techniques for satellite communications. Computer Networks, 173, p.107213.

[70] T. Shapira and Y. Shavitt, "FlowPic: A Generic Representation for Encrypted Traffic Classification and Applications Identification," in IEEE Transactions on Network and Service Management, vol. 18, no. 2, pp. 1218-1232, June 2021, doi: 10.1109/TNSM.2021.3071441.

[71] Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2020). A review on ML–based approaches for internet traffic classification. Annals of Telecommunications, 75(11), 673-710.

[72] Aceto, G., Ciuonzo, D., Montieri, A. and Pescapè, A., 2019. MIMETIC: Mobile encrypted traffic classification using multimodal deep learning. Computer Networks, 165, p.106944.

[73] Gu, C., Zhang, S., Xue, X., & Huang, H. (2011). Online wireless mesh network traffic classification using ML. Journal of Computational Information Systems, 7(5), 1524-1532.

[74] Rezaei, S. and Liu, X., 2019. Deep learning for encrypted traffic classification: An overview. IEEE communications magazine, 57(5), pp.76-81.

[75] D'Angelo, G., & Palmieri, F. (2021). Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. Journal of Network and Computer Applications, 173, 102890.

[76] Thupae, R., Isong, B., Gasela, N. and Abu-Mahfouz, A.M., 2018, October. Machine learning techniques for traffic identification and classifiacation in SDWSN: A survey. In IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society (pp. 4645-4650). IEEE.

[77] Takyi, K., Bagga, A. and Goopta, P., 2018, August. Clustering techniques for traffic classification: a comprehensive review. In 2018 7th International Conference on

Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 224-230). IEEE.

[78] Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.L.A., Elkhatib, Y., Hussain, A. and Al-Fuqaha, A., 2019. Unsupervised machine learning for networking: Techniques, applications and research challenges. IEEE Access, 7, pp.65579-65615.

[79] Kim, J., Sim, A., Tierney, B., Suh, S., & Kim, I. (2019). Multivariate network traffic analysis using clustered patterns. Computing, 101(4), 339-361.

[80] Erman, J., Mahanti, A., Arlitt, M., & Williamson, C. (2007, May). Identifying and discriminating between web and peer-to-peer traffic in the network core. In Proceedings of the 16th international conference on World Wide Web (pp. 883-892).

[81] Iliyasu, A.S. and Deng, H., 2019. Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks. IEEE Access, 8, pp.118-126.

[82] Shaikh, Z. A., & Harkut, D. G. (2015, April). A novel framework for network traffic classification using unknown flow detection. In 2015 Fifth International conference on communication systems and network technologies (pp. 116-121). IEEE.

[83] Zhang, J., Chen, X., Xiang, Y., & Zhou, W. (2013, November). Zero-day traffic identification. In International Symposium on Cyberspace Safety and Security (pp. 213-227). Springer, Cham.

[84] Glennan, T., Leckie, C., & Erfani, S. M. (2016, July). Improved classification of known and unknown network traffic flows using semi-supervised ML. In Australasian conference on information security and privacy (pp. 493-501). Springer, Cham.

[85] Ran, J., Kong, X., Lin, G., Yuan, D., & Hu, H. (2017, December). A self-adaptive network traffic classification system with unknown flow detection. In 2017 3rd IEEE International Conference on Computer and Communications (ICCC) (pp. 1215-1220). IEEE.

[86] M. Wang, Y. Cui, X. Wang, S. Xiao and J. Jiang, "Machine Learning for Networking: Workflow, Advances and Opportunities," in IEEE Network, vol. 32, no. 2, pp. 92-99, March-April 2018, doi: 10.1109/MNET.2017.1700200.

[87] Zhang, J., Chen, X., Xiang, Y., Zhou, W., & Wu, J. (2014). Robust network traffic classification. IEEE/ACM transactions on networking, 23(4), 1257-1270.

[88] S. Khalid, T. Khalil and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," 2014 Science and Information Conference, 2014, pp. 372-378, doi: 10.1109/SAI.2014.6918213.

[89] Jiang, J., Sekar, V., Milner, H., Shepherd, D., Stoica, I. and Zhang, H., 2016. {CFA}: A practical prediction system for video qoe optimization. In 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16) (pp. 137-150).

[90] Fadlullah, Z.M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T. and Mizutani, K., 2017. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. IEEE Communications Surveys & Tutorials, 19(4), pp.2432-2455.

[91] M. Iliofotou, H. Kim, M. Faloutsos, M. Mitzenmacher, P. Pappu and G. Varghese, "Graph-Based P2P Traffic Classification at the Internet Backbone," IEEE INFOCOM Workshops 2009, 2009, pp. 1-6, doi: 10.1109/INFCOMW.2009.5072151.

[92] Iwana, B.K. and Uchida, S., 2020. Time series classification using local distance-based features in multi-modal fusion networks. Pattern Recognition, 97, p.107024.

[93] Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. Expert Systems with Applications, 73, 220-239.

[94] Ansari, G., Ahmad, T. and Doja, M.N., 2019. Hybrid Filter–Wrapper feature selection method for sentiment classification. Arabian Journal for Science and Engineering, 44(11), pp.9191-9208.

[95] Shang, W., Huang, H., Zhu, H., Lin, Y., Qu, Y., & Wang, Z. (2007). A novel feature selection algorithm for text categorization. Expert Systems with Applications, 33(1), 1-5.

[96] Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence, 27(8), 1226-1238.

[97] Raileanu, L. E., & Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. Annals of Mathematics and Artificial Intelligence, 41(1), 77-93.

[98] Karegowda, A. G., Manjunath, A. S., & Jayaram, M. A. (2010). Comparative study of attribute selection using gain ratio and correlation based feature selection. International Journal of Information Technology and Knowledge Management, 2(2), 271-277.

[99] Klepaczko, A., & Materka, A. (2010, June). Combining evolutionary and sequential search strategies for unsupervised feature selection. In International Conference on Artificial Intelligence and Soft Computing (pp. 149-156). Springer, Berlin, Heidelberg.

[100] Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. Pattern recognition, 36(2), 451-461.

[101] Yang, M. S., Lai, C. Y., & Lin, C. Y. (2012). A robust EM clustering algorithm for Gaussian mixture models. Pattern Recognition, 45(11), 3950-3961.

[102] Khan, K., Rehman, S. U., Aziz, K., Fong, S., & Sarasvady, S. (2014, February). DBSCAN: Past, present and future. In The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014) (pp. 232-238). IEEE.

[103] Cheng, Y., Xu, Y., Zhong, H. and Liu, Y., 2019, October. HS-TCN: A semi-supervised hierarchical stacking temporal convolutional network for anomaly

detection in IoT. In 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (pp. 1-7). IEEE.

[104] Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.

[105] Lipton, Z.C., Elkan, C. and Naryanaswamy, B., 2014, September. Optimal thresholding of classifiers to maximize F1 measure. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 225-239). Springer, Berlin, Heidelberg.

[106] Powers, D. M. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061

[107] Zhang, C., Pan, X., Li, H., Gardiner, A., Sargent, I., Hare, J. and Atkinson, P.M., 2018. A hybrid MLP-CNN classifier for very fine resolution remotely sensed image classification. ISPRS Journal of Photogrammetry and Remote Sensing, 140, pp.133-144.

[108] Heidari, A. A., Faris, H., Mirjalili, S., Aljarah, I., & Mafarja, M. (2020). Ant lion optimizer: theory, literature review, and application in multi-layer perceptron neural networks. Nature-inspired optimizers, 23-46.

[109] Rosa, João PS, et al. Using artificial neural networks for analog integrated circuit design automation. Vol. 1. Berlin: Springer, 2020.

[110] Kulin, M., Kazaz, T., De Poorter, E., & Moerman, I. (2021). A survey on machine learning-based performance improvement of wireless networks: PHY, MAC and network layer. Electronics, 10(3), 318.

[111] Jiang, Hui. Machine Learning Fundamentals: A Concise Introduction. Cambridge University Press, 2021.

[112] Alsharef, Ahmad, et al. "Review of ML and AutoML solutions to forecast time-series data." Archives of Computational Methods in Engineering 29.7 (2022): 5297-5311.

[113] Hasebrook, Niklas, et al. "Why Do Machine Learning Practitioners Still Use Manual Tuning? A Qualitative Study." arXiv preprint arXiv:2203.01717 (2022).

[114] Tu, Huy, and Vivek Nair. "Is one hyperparameter optimizer enough?." Proceedings of the 4th ACM SIGSOFT international workshop on software analytics. 2018.

[115] El-Nasr, Magy Seif, et al. Game Data Science. Oxford University Press, 2021.

[116] Sun, Y., Zhang, H., Zhao, T., Zou, Z., Shen, B. and Yang, L., 2020. A new convolutional neural network with random forest method for hydrogen sensor fault diagnosis. IEEE Access, 8, pp.85421-85430.

[117] Mirtaheri, Seyedeh Leili, and Reza Shahbazian. Machine Learning: Theory to Applications. CRC Press, 2022.

[118] Chakraborty, Sanjay, SK Hafizul Islam, and Debabrata Samanta. Data Classification and Incremental Clustering in Data Mining and Machine Learning. Springer, 2022.

[119] Alzubaidi, Laith, et al. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions." Journal of big Data 8.1 (2021): 1-74.

[120] Janiesch, Christian, Patrick Zschech, and Kai Heinrich. "Machine learning and deep learning." Electronic Markets 31.3 (2021): 685-695.

[121] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354-377.

[122] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[123] Yin, C., Zhu, Y., Fei, J. and He, X., 2017. A deep learning approach for intrusion detection using recurrent neural networks. Ieee Access, 5, pp.21954-21961.

[124] Nasir, Vahid, and Farrokh Sassani. "A review on deep learning in machining and tool monitoring: methods, opportunities, and challenges." The International Journal of Advanced Manufacturing Technology 115.9 (2021): 2683-2709.

[125] Fergus, Paul, and Carl Chalmers. Applied Deep Learning: Tools, Techniques, and Implementation. Springer Nature, 2022.

[126] Aceto, G., Ciuonzo, D., Montieri, A., & Pescapé, A. (2019). Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. IEEE Transactions on Network and Service Management, 16(2), 445-458.

[127] Shbair, W. M., Cholez, T., Francois, J., & Chrisment, I. (2016, April). A multi-level framework to identify https services. In NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium (pp. 240-248). IEEE.

[128] Aceto, G., Ciuonzo, D., Montieri, A., & Pescapé, A. (2018). Multi-classification approaches for classifying mobile app traffic. Journal of Network and Computer Applications, 103, 131-145.

[129] Y. Fu, H. Xiong, X. Lu, J. Yang and C. Chen, "Service Usage Classification with Encrypted Internet Traffic in Mobile Messaging Apps," in IEEE Transactions on Mobile Computing, vol. 15, no. 11, pp. 2851-2864, 1 Nov. 2016, doi: 10.1109/TMC.2016.2516020.

[130] Taylor, V. F., Spolaor, R., Conti, M., & Martinovic, I. (2016, March). Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 439-454). IEEE.

[131] Alan, H. F., & Kaur, J. (2016, July). Can Android applications be identified using only TCP/IP headers of their launch time traffic?. In Proceedings of the 9th ACM conference on security & privacy in wireless and mobile networks (pp. 61-66).

[132] Zhao M, Zhang T, Ge F, Yuan Z (2012) RobotDroid: a lightweight malware detection framework on smartphones. J Netw 7(4):715.

[133] Stöber T, Frank M, Schmitt J, Martinovic I. Who do you sync you are? Smartphone fingerprinting via application behaviour. InProceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks 2013 Apr 17 (p. 8). ACM.

[134] V. F. Taylor, R. Spolaor, M. Conti and I. Martinovic, "Robust Smartphone App Identification via Encrypted Network Traffic Analysis," in IEEE Transactions on Information Forensics and Security, vol. 13, no. 1, pp. 63-78, Jan. 2018, doi: 10.1109/TIFS.2017.2737970.

[135] Erman, J., Mahanti, A., Arlitt, M., Cohen, I., & Williamson, C. (2007). Offline/realtime traffic classification using semi-supervised learning. Performance Evaluation, 64(9-12), 1194-1213.

[136] Erman, J., Arlitt, M., & Mahanti, A. (2006, September). Traffic classification using clustering algorithms. In Proceedings of the 2006 SIGCOMM workshop on Mining network data (pp. 281-286).

[137] Williams, N., Zander, S., & Armitage, G. (2006). A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. ACM SIGCOMM Computer Communication Review, 36(5), 5-16.

[138] Bernaille L, Teuxeira R, Akodkenous I, Soule A, Slamatian K. Traffic classification on the fly. ACM SIGCOMM Computer Communication Review 36: 23–26, 2006.

[139] Chen, Y. C., Li, Y. J., Tseng, A., & Lin, T. (2017, October). Deep learning for malicious flow detection. In 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC) (pp. 1-7). IEEE.

[140] Wang, W., Zhu, M., Zeng, X., Ye, X., & Sheng, Y. (2017, January). Malware traffic classification using convolutional neural network for representation learning. In 2017 International Conference on Information Networking (ICOIN) (pp. 712-717). IEEE.

[141] J. Han, J. Pei and M. Kamber, Data Mining: Concepts and Techniques, Amsterdam, The Netherlands:Elsevier, 2011.

[142] M. Anand, E.B. Nightingale and J. Flinn, "Self-tuning wireless network power management" in Proc. of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM '03), San Diego, CA, Sept. 2003, pp. 176–189.

[143] H. Peng, F. Long and C. Ding, "Feature selection based on mutual information criteria of max-dependency max-relevance and min-redundancy", IEEE Trans. Pattern Anal. Mach. Intell., vol. 27, no. 8, pp. 1226-1238, Aug. 2005.

[144] J. Li et al., "Feature selection: A data perspective", ACM Comput. Surv., vol. 50, no. 6, pp. 94, Jan. 2017.

[145] L. Morán-Fernández, V. Bolón-Canedo and A. Alonso-Betanzos, "Centralized vs. distributed feature selection methods based on data complexity measures", Knowl.-Based Syst., vol. 117, pp. 27-45, Feb. 2017.

[146] "Weka Website" [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/.

[147] L. Verde, G. De Pietro and G. Sannino, "Voice disorder identification by using machine learning techniques", IEEE Access, vol. 6, pp. 16246-16255, 2018.

[148] S. B. Sakri, N. B. A. Rashid and Z. M. Zain, "Particle swarm optimization feature selection for breast cancer recurrence prediction", IEEE Access, vol. 6, pp. 29637-29647, 2018.

[149] R. Kohavi et al., "A study of cross-validation and bootstrap for accuracy estimation and model selection", Proc. Int. Joint Conf. AI, vol. 14, no. 2, pp. 1137-1145, Aug. 1995.

[150] F. A. Narudin, A. Feizollah, N. B. Anuar and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection", Soft Comput., vol. 20, no. 1, pp. 343-357, 2016.

[151] Y. Li, X. Zhang and K. L. Yeung, "A novel delayed wakeup scheme for efficient power management in infrastructure-based IEEE 802.11 WLANs," in 2015 IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, LA, USA, 2015, pp. 1338-1343.

[152] Y. Xie, X. Luo and R. K. C. Chang, "Centralizing the power saving mode for 802.11 infrastructure networks" in Energy Technology and Management, London, U.K.:IntechOpen, pp. 3-24, 2011, [online] Available: https://www.intechopen.com/chapters/20631.

[153] D. Jung, R. Kim, and H. Lim, "Power-saving strategy for balancing energy and delay performance in WLANs," Computer Communications, vol. 50, pp.3-9, 2014.

[154] F. Wu et al., "Cutting Down Idle Listening Time: A NDN-Enabled Power Saving Mode Design for WLAN," in ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6.

[155] S. Sur, T. Wei, and X. Zhang, "Bringing multi-antenna gain to energy-constrained wireless devices," in Proc. of the 14th International Conference on Information Processing in Sensor Networks, pp. 25-36, 2015.

[156] Jiang, N., Vuran, M. C., Wei, S., & Xu, L. (2021, June). QoS-Aware Network Energy Optimization for Danmu Video Streaming in WiFi Networks. In 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) (pp. 1-10). IEEE.

[157] Liu, Z., Qin, X., & Wei, G. (2021, June). GreenAP: An Energy-Saving Protocol for Mobile Access Points. In International Conference on Wireless Algorithms, Systems, and Applications (pp. 553-565). Springer, Cham.

[158] Asad, M., & Qaisar, S. (2022). Energy Efficient QoS Based Access Point Selection in Hybrid WiFi and LiFi IoT Networks. IEEE Transactions on Green Communications and Networking.

[159] Luo, Y., & Chin, K. W. (2021). An Energy Efficient Channel Bonding and Transmit Power Control Approach for WiFi Networks. IEEE Transactions on Vehicular Technology, 70(8), 8251-8263.

[160] Venkateswaran, S. K., Tai, C. L., Ben-Yehezkel, Y., Alpert, Y., & Sivakumar, R. (2021, November). Extending Battery Life for Wi-Fi-Based IoT Devices: Modeling, Strategies, and Algorithm. In Proceedings of the 19th ACM International Symposium on Mobility Management and Wireless Access (pp. 147-156).

[161] Luo, Y., & Chin, K. W. (2021). Learning to Charge RF-Energy Harvesting Devices in WiFi Networks. IEEE Systems Journal, 15 (4), 5516-5525.

[162] Ko, Jaejun, Young-June Choi, and Rajib Paul. "Computation offloading technique for energy efficiency of smart devices." Journal of Cloud Computing 10.1 (2021): 1-14. Springer.

[163] Sheth, J., Miremadi, C., Dezfouli, A., & Dezfouli, B. (2022). EAPS: Edge-assisted predictive sleep scheduling for 802.11 IoT stations. IEEE Systems Journal.

[164] Yang, C., Lee, J., & Bahk, S. (2021, March). Target wake time scheduling strategies for uplink transmission in IEEE 802.11 ax networks. In 2021 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-6). IEEE.

[165] Zhang, M., Zhu, Y. H., & Liu, Y. (2021, March). Throughput Aware Users Allocation Scheme for Coexistence of the LTE system and 802.11 ax WLANs. In 2021 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-7). IEEE.

[166] Badarla, S. P., & Harigovindan, V. P. (2021). Restricted Access Window-Based Resource Allocation Scheme for Performance Enhancement of IEEE 802.11 ah Multi-Rate IoT Networks. IEEE Access, 9, 136507-136519.

[167] Lu, C., Zhao, Y., Bao, J., Chen, S. L., Liu, J., Chien, C. M., ... & Li, Y. (2021, September). A Highly Efficient Combo Transceiver for 802.11 b/g/n/ax and BT/BLE in 22nm CMOS. In ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC) (pp. 503-506). IEEE.

[168] Aggarwal, S., Ghoshal, M., Banerjee, P., Koutsonikolas, D., & Widmer, J. (2021, May). 802.11 ad in Smartphones: Energy Efficiency, Spatial Reuse, and Impact on Applications. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications (pp. 1-10). IEEE.

[169] Menzies, T., Kocaguneli, E., Turhan, B., Minku, L., & Peters, F. (2015). Sharing data and models in software engineering. Morgan Kaufmann.

[170] Lotte, F. (2008). Study of electroencephalographic signal processing and classification techniques towards the use of brain-computer interfaces in virtual reality applications (Doctoral dissertation, INSA de Rennes).

[171] Somerset, V. S. (2010). Intelligent and Biosensors, Edited by Vernon S. Somerset. Intech, January.

[172] Hassanien, A. E., & Azar, A. A. (2015). Brain-computer interfaces. Switzerland: Springer, 74.

[173] Wang, L., Wang, X. V., Váncza, J., & Kemény, Z. (Eds.). (2021). Advanced Human-Robot Collaboration in Manufacturing. Springer International Publishing.

[174] Boo, Y. L., Stirling, D., Chi, L., Liu, L., Ong, K. L., & Williams, G. (Eds.). (2018). Data Mining: 15th Australasian Conference, AusDM 2017, Melbourne, VIC, Australia, August 19-20, 2017, Revised Selected Papers (Vol. 845). Springer.

[175] Thomas, P., & Suhner, M. C. (2015). A new multilayer perceptron pruning algorithm for classification and regression applications. Neural Processing Letters, 42(2), 437-458.

[176] Marwala, T. (2018). Handbook Of Machine Learning-Volume 1: Foundation Of Artificial Intelligence. World Scientific.

[177] Nelles, O. (2020). Nonlinear system identification: from classical approaches to neural networks, fuzzy models, and gaussian processes. Springer Nature.

[178] Arévalo, A., Niño, J., Hernandez, G., Sandoval, J., León, D., & Aragón, A. (2017, September). Algorithmic Trading Using Deep Neural Networks on High Frequency Data. In Workshop on Engineering Applications (pp. 144-155). Springer, Cham.

[179] Lotte, F., Congedo, M., Lécuyer, A., Lamarche, F., & Arnaldi, B. (2007). A review of classification algorithms for EEG-based brain–computer interfaces. Journal of neural engineering, 4(2), R1.

[180] Swingler, K. (2016). Opening the black box: analysing MLP functionality using walsh functions. In Computational Intelligence (pp. 303-323). Springer, Cham.

[181] Swingler, K. (2016). Mixed Order Hyper Networks for Function Approximation and Optimisation. PhD thesis, Stirling University.

[182] Al Bataineh, A., Kaur, D., & Jalali, S. M. J. (2022). Multi-Layer Perceptron Training Optimization Using Nature Inspired Computing. IEEE Access, 10, 36963-36977.

[183] Haselsteiner, E., & Pfurtscheller, G. (2000). Using time-dependent neural networks for EEG classification. IEEE transactions on rehabilitation engineering, 8(4), 457-463.

[184] Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). The annals of statistics, 26(3), 801-849.

[185] Das, S., Tripathy, D., & Raheja, J. L. (2019). Real-time BCI system design to control arduino based speed controllable robot using EEG. Springer.

[186] Salgado, Cátia M., et al. "Noise versus outliers." Secondary analysis of electronic health records (2016): 163-183.

[187] Hemanth, J., Bestak, R., & Chen, J. I. Z. (Eds.). (2021). Intelligent data communication technologies and internet of things: proceedings of ICICI 2020. Springer Singapore.

[188] Meng, W., & Furnell, S. (2019). Security and Privacy in Social Networks and Big Data. Springer Singapore.

[189] Hu, Li-Yu, et al. "The distance function effect on k-nearest neighbor classification for medical datasets." SpringerPlus 5.1 (2016): 1-9.

[190] Du, Zhi-Gang, et al. "QUasi-Affine TRansformation evolutionary algorithm for feature selection." Advances in Smart Vehicular Technology, Transportation, Communication and Applications. Springer, Singapore, 2022. 147-156.

[191] Brownlee, Jason. "Master Machine Learning Algorithms: Discover How They Work and Implement Them From Scratch, 2016." URL https://books. google. ca/books.

[192] Larose, Daniel T., and Chantal D. Larose. Discovering knowledge in data: an introduction to data mining. Vol. 4. John Wiley & Sons, 2014.

[193] Venugopal, Deepak, Lih-Yuan Deng, and Max Garzon. "Solutions to Data Science Problems." Dimensionality Reduction in Data Science. Springer, Cham, 2022. 29-65.

[194] Zhou, Zhi-Hua. Machine learning. Springer Nature, 2021.

[195] Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Machine Learning: Fundamental Concepts and Algorithms, 2nd Edition, Cambridge University Press, March 2020. ISBN: 978-1108473989.

[196] Awad, Mariette, and Rahul Khanna. Efficient learning machines: theories, concepts, and applications for engineers and system designers. Springer nature, 2015.

[197] Gholami, Raoof, and Nikoo Fakhari. "Support vector machine: principles, parameters, and applications." Handbook of neural computation. Academic Press, 2017. 515-535.

[198] Li, Li. Selected applications of convex optimization. Vol. 103. Berlin, Germany: Springer, 2015.

[199] Flach, Peter. Machine learning: the art and science of algorithms that make sense of data. Cambridge university press, 2012.

[200] Silva, Warley Almeida, and Saulo Moraes Villela. "Improving the one-against-all binary approach for multiclass classification using balancing techniques." Applied Intelligence 51.1 (2021): 396-415.

[201] Abe, Shigeo. Support vector machines for pattern classification. Vol. 2. London: Springer, 2010.

[202] Zhou, Jian, Shuai Huang, and Yingui Qiu. "Optimization of random forest through the use of MVO, GWO and MFO in evaluating the stability of underground entry-type excavations." Tunnelling and Underground Space Technology 124 (2022): 104494.

[203] Aridas, Christos K., et al. "Random resampling in the one-versus-all strategy for handling multi-class problems." International Conference on Engineering Applications of Neural Networks. Springer, Cham, 2017.

[204] A. Géron, Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow. Sebastopol, CA, USA: O'reilly, 2019.

[205] Caballé, Santi, et al., eds. Intelligent systems and learning data analytics in online education. Academic Press, 2021.

[206] Chandak, Aniket, Wendy Lee, and Mark Stamp. "A comparison of Word2Vec, HMM2Vec, and PCA2Vec for malware classification." Malware Analysis Using Artificial Intelligence and Deep Learning. Springer, Cham, 2021. 287-320.

[207] Samanta, D., Islam, S.H., Chilamkurti, N., & Hammoudeh, M. (Eds.). (2022). Data Analytics, Computational Statistics, and Operations Research for Engineers: Methodologies and Applications (1st ed.). CRC Press.

[208] Freitas, Alex A. "Automated machine learning for studying the trade-off between predictive accuracy and interpretability." International cross-domain conference for machine learning and knowledge extraction. Springer, Cham, 2019.

[209] Ismail, Muhammad, Changjing Shang, and Qiang Shen. "Towards a Framework for Interpretation of CNN Results with ANFIS." UK Workshop on Computational Intelligence. Springer, Cham, 2021.

[210] Gou, Jianping, et al. "A generalized mean distance-based k-nearest neighbor classifier." Expert Systems with Applications 115 (2019): 356-372.

[211] Tanwar, Sudeep, Sudhanshu Tyagi, and Neeraj Kumar, eds. Multimedia big data computing for IoT applications: concepts, paradigms and solutions. Vol. 163. Springer, 2019.

[212] A. V. Joshi, Machine Learning and Artificial Intelligence, 2nd Edition, Cham, Switzerland: Springer, 2023. doi: https://doi.org/10.1007/978-3-031-12282-8.

[213] Dahouda, Mwamba Kasongo, and Inwhee Joe. "A deep-learned embedding technique for categorical features encoding." *IEEE Access* 9 (2021): 114381-114391.

[214] N. Dey and A.S. Ashour, Classification and Clustering in Biomedical Signal Processing, 2016.

[215] Valdez-Valenzuela, Eric, Angel Kuri-Morales, and Helena Gomez-Adorno. "CESAMMO: Categorical Encoding by Statistical Applied Multivariable

Modeling." *Mexican International Conference on Artificial Intelligence*. Springer, Cham, 2022.

[216] Gupta, Richa, et al. "Transformation of Medical Imaging Using Artificial Intelligence: Its Impact and Challenges with Future Opportunities." Soft Computing: Theories and Applications. Springer, Singapore, 2022. 201-212.

[217] Kantardzic, Mehmed. Data mining: concepts, models, methods, and algorithms. John Wiley & Sons, 2019.

[218] Khalid, Nur Hidayah Mohd, Amelia Ritahani Ismail, and Normaziah A. Aziz. "Interpretation of Machine Learning Model Using Medical Record Visual Analytics." Proceedings of the 8th International Conference on Computational Science and Technology. Springer, Singapore, 2022.

[219] Zhang, Aston, et al. "Dive into deep learning." arXiv preprint arXiv:2106.11342 (2022).

[230] Mantovani, Rafael G., et al. "A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves SVM classifiers." Information Sciences 501 (2019): 193-221.

[231] Moon, Jihoon, et al. "A short-term electric load forecasting scheme using 2-stage predictive analytics." 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE, 2018.

[232] Khalid, Rabiya, and Nadeem Javaid. "A survey on hyperparameters optimization algorithms of forecasting models in smart grid." Sustainable Cities and Society 61 (2020): 102275.

[233] Chen, Yutian, et al. "Bayesian optimization in alphago." arXiv preprint arXiv:1812.06855 (2018).

[234] Zhang, Baohe, et al. "On the importance of hyperparameter optimization for model-based reinforcement learning." International Conference on Artificial Intelligence and Statistics. PMLR, 2021.

[235] Kadra, Arlind, et al. "Well-tuned simple nets excel on tabular datasets." Advances in neural information processing systems 34 (2021): 23928-23941.

[236] Bouthillier, Xavier, and Gaël Varoquaux. Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020. Diss. Inria Saclay Ile de France, 2020.

[237] Blom, K. van der, Serban, A. C., Hoos, H. H., & Visser, J. M. W. (2021). AutoML adoption in ML software. 8Th Icml Workshop On Automated Machine Learning. Retrieved from https://hdl.handle.net/1887/3277270.

[238] Mantovani, Rafael Gomes, et al. "Rethinking default values: a low cost and efficient strategy to define hyperparameters." arXiv preprint arXiv:2008.00025 (2020).

[239] Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." Neurocomputing 415 (2020): 295-316.

[240] Mbiki, Sarah, et al. "Classifying changes in LN-18 glial cell morphology: a supervised machine learning approach to analyzing cell microscopy data via FIJI and WEKA." Medical & Biological Engineering & Computing 58 (2020): 1419-1430.

[241] Chou, Jui-Sheng, Dillon-Brandon Fleshman, and Dinh-Nhat Truong. "Comparison of machine learning models to provide preliminary forecasts of real estate prices." Journal of Housing and the Built Environment (2022): 1-36.

[242] Idri, Ali, et al. "Assessing the impact of parameters tuning in ensemble based breast Cancer classification." Health and Technology 10 (2020): 1239-1255.

[243] Sung, Sheng-Feng, Chia-Yi Lin, and Ya-Han Hu. "EMR-based phenotyping of ischemic stroke using supervised machine learning and text mining techniques." IEEE journal of biomedical and health informatics 24.10 (2020): 2922-2931.

[244] Carlin, Domhnall, Philip O'Kane, and Sakir Sezer. "A cost analysis of machine learning using dynamic runtime opcodes for malware detection." Computers & Security 85 (2019): 138-155.

[245] Alis, D. E. N. İ. Z., et al. "The diagnostic value of quantitative texture analysis of conventional MRI sequences using artificial neural networks in grading gliomas." Clinical radiology 75.5 (2020): 351-357.

[246] Li, Tengyue, et al. "Empowering multi-class medical data classification by Group-of-Single-Class-predictors and transfer optimization: Cases of structured dataset by machine learning and radiological images by deep learning." Future Generation Computer Systems 133 (2022): 10-22.

[247] Nieto, Paulino José García, et al. "Forecast of the higher heating value based on proximate analysis by using support vector machines and multilayer perceptron in bioenergy resources." Fuel 317 (2022): 122824.

[248] Bhojani, Shital H., and Nirav Bhatt. "Wheat crop yield prediction using new activation functions in neural network." Neural Computing and Applications 32 (2020): 13941-13951.

[249] Kumar, J. Ashok, and S. Abirami. "Ensemble application of bidirectional LSTM and GRU for aspect category detection with imbalanced data." Neural Computing and Applications 33.21 (2021): 14603-14621.

[250] Zhou, Gordon, Amir Etemadi, and Austin Mardon. "Machine learning-based cost predictive model for better operating expenditure estimations of US light rail transit projects." Journal of Public Transportation 24 (2022): 100031.

[251] Altuve, Miguel, Antonio J. Alvarez, and Erika Severeyn. "Multiclass classification of metabolic conditions using fasting plasma levels of glucose and insulin." Health and Technology 11 (2021): 953-962.

[252] Carballo-Meilan, Ara, et al. "Meta-analysis of vaterite secondary data revealed the synthesis conditions for polymorphic control." Chemical Engineering Research and Design 188 (2022): 668-680.

[253] Rao, U. Mohan, et al. "Identification and application of machine learning algorithms for transformer dissolved gas analysis." IEEE Transactions on Dielectrics and Electrical Insulation 28.5 (2021): 1828-1835.

[254] Shawki, N., et al. "On automating hyperparameter optimization for deep learning applications." 2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB). IEEE, 2021.

[255] Mukherjee, Himadri, et al. "Automatic lung health screening using respiratory sounds." Journal of Medical Systems 45 (2021): 1-9.

[256] Ghatasheh, Nazeeh, et al. "Cost-sensitive ensemble methods for bankruptcy prediction in a highly imbalanced data distribution: A real case from the Spanish market." Progress in Artificial Intelligence 9 (2020): 361-375.

[257] Kumaravel, A., and T. Vijayan. "Comparing cost sensitive classifiers by the false-positive to false-negative ratio in diagnostic studies." Expert Systems with Applications 227 (2023): 120303.