A Novel Framework for Analysis of Big Data

A thesis

submitted in fulfillment of the requirements

for the award of degree of

Doctor of Philosophy

Submitted by

Deepak Gupta

(Reg. No.: 951203005)

Under the guidance & supervision of

Dr. Rinkle Rani

(Associate Professor, Department of Computer Science & Engineering)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY

PATIALA – 147004 (INDIA).

July 2019

Table of Contents

List of Figures ..... *vi*

List of Tables ..... *viii*

Abstract ..... *xi*

Chapter 1: Introduction *1*

1.1 Big Data Deluge **1**

1.2 Big Data Evolution **4**

1.2.1 Brief History 4

1.2.2 Defining Big Data 4

1.2.3 Key Characteristics 5

1.2.4 Big Data Classification 7

1.3 A Bibliometric Study of Research Publications in Academia and Industry **10**

1.3.1 Methodology 10

1.3.2 Key Findings 11

1.3.3 Top Journals and Magazines 14

1.3.4 Title Words and Author-Assigned Keywords Statistics 14

1.4 Open Source Big Data Stream Processing Frameworks **17**

1.4.1 Apache Hadoop 17

1.4.2 Apache Spark 18

1.4.3 Apache Storm 18

1.4.4 Apache Samza 18

1.4.5 Apache Flink 22

1.4.6 Apache Apex 22

1.5 Major Research Challenges **23**

1.5.1 Data Acquisition and Metadata 24

1.5.2 Data Quality 24

1.5.3 Data Storage 24

1.5.4 Data Sharing and Transfer 25

1.5.5 Scalability 25

1.5.6 Data Analysis 26

1.5.7 Data Querying and Indexing 27

1.5.8 Data Uncertainty 28

1.5.9 Data Privacy, Security and Ethics 28
1.5.10 Data Visualization 28
1.6 Thesis Outline 29
1.7 Summary 30

Chapter 2: Background and Related Research Work 32
2.1 Introduction 32
2.2 Big Data Analytics 34
2.3 Big Data Analytics, Machine Learning, and Deep Learning in Cybersecurity 36
2.3.1 Scalable Machine Learning Libraries 38
2.3.2 Open Source Deep Learning Frameworks 40
2.3.2.1 Tensorflow 40
2.3.2.2 Keras 40
2.3.2.3 PyTorch 43
2.3.2.4 Apache MXNet 43
2.3.2.5 Microsoft Cognitive Toolkit (CNTK) 44
2.3.2.6 Chainer 44
2.4 Big Data Security Analytics 44
2.4.1 Key Features 45
2.4.2 Use Cases 48
2.4.3 High-Level Conceptual Architecture 49
2.4.4 Examples from Literature 50
2.4.5 Existing Solutions 53
2.4.5.1 Apache Metron 54
2.4.5.2 Apache Spot 54
2.4.5.3 IBM QRadar 54
2.4.5.4 Splunk Enterprise Security (ES) 55
2.4.5.5 LogRhythm 55
2.4.5.6 RSA Security Analytics 56
2.4.5.7 SNYPR 56
2.4.5.8 E8 Security 56
2.5 Malware Classification: A Case Study 59
2.5.1 Overview of Malware Analysis 60
2.5.2 Malware Detection Methods 61
2.5.3 Malware Classification using Machine Learning Methods 62
2.5.4 Malware Classification using Deep Learning Methods 66
2.6 Summary 67

Chapter 3: Problem Formulation *68*
3.1 Research Gaps **68**
3.2 Problem Statement **69**
3.3 Research Objectives **70**
3.4 Research Methodology **70**
3.5 Summary **72**

Chapter 4: Architecture for Big Data Analytics *73*
4.1 introduction **73**
4.2 Proposed Architecture **74**
4.2.1 Data Preparation 75
4.2.1.1 Data Cleaning and Labelling 77
4.2.2 Big Data Framework 78
4.2.3 Functional Flow 78
4.3 Malware Trends Analysis **80**
4.3.1 Experimental Setup 81
4.3.2 Malware Binaries Size 81
4.3.3 Commonly used Malware Packers 84
4.3.4 Network Activities 87
4.3.5 Windows Registry Activities 88
4.4 Discussion and Key Findings **90**
4.5 Summary **91**

Chapter 5: Big Data Framework for Zero-Day Malware Classification *93*
5.1 Introduction **93**
5.2 Preliminaries **94**
5.2.1 Naïve Bayes 94
5.2.2 Support Vector Machine 95
5.2.3 Random Forest 95
5.3 Proposed Architecture for Zero-Day Malware Detection **95**
5.3.1 Feature Extraction and Aggregation 96
5.3.1.1 File Metadata 97
5.3.1.2 File Size 97
5.3.1.3 Packer Detection 97
5.3.1.4 Sections' Information 98
5.3.1.5 Dynamically Linked Libraries (DLLs) 98
5.3.1.6 Dropped Files 98

5.3.1.7 Windows API Calls 99
5.3.1.8 Mutex Operations 99
5.3.1.9 File Activities 99
5.3.1.10 Registry Activities 99
5.3.1.11 Network Activities 101
5.3.1.12 Process Activities 101
5.3.2 Big Data Tools 101
5.3.3 Functional Flow 101
5.4 Malware Feature Impact Analysis 102
5.5 Evaluation 104
5.5.1 Dataset 104
5.5.2 Experimental Setup 106
5.5.3 Evaluation Parameters 107
5.5.4 Cross Validation 109
5.5.5 Experimental Results 109
5.6 Discussion 111
5.7 Summary 111

Chapter 6: Improving Malware Detection using Big Data and Ensemble Learning 112
6.1 Ensemble Learning – An Overview 112
6.1.1 Data Level 113
6.1.2 Feature Level 113
6.1.3 Classifier Level 113
6.1.4 Combination Level 113
6.1.4.1 Stacking 113
6.1.4.2 Voting 114
6.1.4.3 Ensemble Selection 114
6.2 Proposed Methodology 115
6.2.1 Malware Classification Using Base Classifiers 115
6.2.1.1 K-Nearest Neighbour 117
6.2.1.2 Decision Table 117
6.2.2 Proposed Schemes 117
6.2.2.1 Scheme I 117
a) Average Accuracy (AA) 118
b) Class Accuracy Differential (CAD) 118
c) Ranked Aggregate per Class Accuracies Based Approach (RACA) 119
d) Ranked Aggregate of Average Accuracy and Class Differential (RACD) 120
6.2.2.2 Scheme II 121

6.3 Evaluation **121**
6.3.1 Dataset 121
6.3.2 Experimental Environment 122
6.3.3 Evaluation Results 122
6.4 Discussion **128**
6.5 Summary **129**

Chapter 7: Malware Classification using Big Data and Deep Neural Network ***130***
7.1 Introduction **130**
7.2 Methodology **131**
7.2.1 Designing Neural Network 133
7.2.2.1 Parameter Initialization 133
7.2.2.2 Selecting Hidden Layers and Neurons 135
7.2.2.3 Activation Functions 135
7.2.2.4 Regularization - Handling overfitting 136
7.2.2.5 Loss Function 137
7.3 Evaluation **138**
7.3.1 Dataset 138
7.3.2 Experimental Environment 138
7.3.3 Training of Deep Neural Network 139
7.3.4 Results and Discussion 140
7.3.4.1 Hyperparameter Tuning 140
7.3.4.2 Evaluation of DNNs and Comparison with Best Base Classifier 141
7.4 Discussion **143**
7.5 Summary **144**

Chapter 8: Conclusion and Future Scope ***145***
8.1 Conclusion **145**
8.2 Research Contributions **148**
8.3 Future Scope **148**

References ***150***
List of Publications ***167***

List of Figures

Figure	Description	Page
1.1	The data analysis pipeline pertaining to bibliometric study of research publications	12
1.2	Distribution of research publications pertaining to big data published from 2000 to 2017	13
1.3	Google worldwide search interest starting from 2010 to 2017 for big data processing frameworks	23
2.1	Types of data analytics	35
2.2	Big data analytics pipeline	35
2.3	Distributed machine learning paradigm along with big data processing engines and their machine learning libraries	39
2.4	A high-level conceptual architecture of big data security analytics	51
2.5	Types of malware analysis	61
3.1	Workflow of the research methodology	71
4.1	Block diagram of proposed architecture for malware trends analysis at large scale	76
4.2	Year-wise distribution of malware samples in the dataset	77
4.3	Functional flow of malware trends analysis	79
4.4	Malware binaries size vs. percentage number of samples	82
4.5	Trend line of mean size of malicious samples	84
4.6	Packers used vs. percentage number of malware binaries	85
4.7	Trend line of packed malware binaries	86
4.8	Trend line of various packers used by malicious samples	86
4.9	Percentage of malware samples showing network behaviour	87
4.10	Usage of HTTP, SMTP, and IRC in malware samples	88
4.11	Comparison of registry activities	89
5.1	Block diagram of proposed architecture for zero-day malware detection	96
5.2	Big data framework for malware classification	102

5.3	Relative importance of malware features	104
5.4	Top 15 malware families in dataset	105
5.5	Number of malware samples (in dataset) submitted to VirusTotal in the years between 2010 and 2016	106
5.6	FPR/FNR comparison of classifiers	110
5.7	Accuracy comparison of classifiers	110
6.1	Proposed methodology for detecting zero day malware	116
6.2	Comparison of base classifiers on the basis of FPR/FNR	123
6.3	Comparison of base classifiers on the basis of accuracy	124
6.4	Comparison of base classifiers on the basis of MCC	124
6.5	Comparison of best base classifier, traditional and proposed ensemble methods using FPR/FNR	127
6.6	Comparison of best base classifier, tradition ensemble and proposed ensemble methods using accuracy	127
6.7	Comparison of base classifiers, tradition ensemble, and proposed ensemble methods using MCC	128
7.1	High-level overview of proposed model	132
7.2	Schematic diagram of proposed DNN. The layers are separated by batch normalization and dropout	134
7.3	Comparison of best base classifier and DNN using FPR/FNR	142
7.4	Comparison of best base classifier and DNN using accuracy	142
7.5	Comparison of best base classifier and DNN using MCC	143

List of Tables

Table	Description	Page
1.1	Popular big data definitions	5
1.2	Big data classification	8-9
1.3	Top 10 journals and magazines with big data publications	15
1.4	Top 30 frequently used words in the title of research publications pertaining to big data	16
1.5	Top 30 commonly used author-assigned keywords	16
1.6	Comparison between Apache Hadoop and Apache Spark platforms.	19
1.7	Comparison of open source big data stream processing frameworks.	20-21
1.8	Various categories of NoSQL databases	26
2.1	Comparison of popular open-source deep learning frameworks	41-42
2.2	Difference between traditional SIEM and BDSA	47
2.3	Comparison of big data security analytics frameworks	57-58
4.1	Experimental environment	81
4.2	Five-number summary (in kilobytes) of malware samples	83
5.1	Selected features and their values	100
5.2	Malware features' dimensions with corresponding weight	103
5.3	Experimental environment	106
5.4	Table of confusion or confusion matrix for malware class	107
5.5	Confusion matrix fields along with their description	107
5.6	Classification results using 10-fold cross validation	110
6.1	Experimental environment	122
6.2	Classification results using base classifiers	123
6.3	Rank and weight of different classifiers using ranking algorithms	125
6.4	Comparison of traditional ensemble methods with the proposed ensemble methods	126
7.1	Experimental environment	139
7.2	Hyperparameter – possible and best values, for model configuration.	141
7.3	Classification results using base classifiers and DNN	141

Certificate

I hereby certify that the work which is presented in this thesis entitled **'A Novel Framework for Analysis of Big Data'**, in fulfillment of the requirements for the award of degree of '**Doctor of Philosophy'**, is submitted in the Department of Computer Science & Engineering, Thapar Institute of Engineering & Technology, Patiala (India). This work is an authentic record of my own research work carried out under the supervision of **Dr. Rinkle Rani** and refers other research works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Deepak Gupta
(Reg. No.: 951203005)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Dr. Rinkle Rani
Associate Professor
Department of Computer Science & Engineering,
Thapar Institute of Engineering & Technology, Patiala (India).

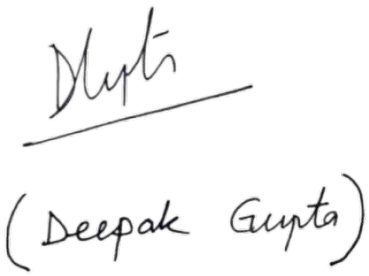
Acknowledgement

Life is a wonderful journey consisting of many milestones. You are bound to meet many noble souls who make this journey more beautiful and memorable. Very often, we either fail to recognise or even forget to appreciate their contributions. As I finish another milestone, I acknowledge the contributions of such people who made it all happen and express my sincere gratitude to all of them.

I would like to thank my supervisor, Dr. Rinkle Rani, for her kindness and guidance throughout the tenure. It is only due to her encouragement, patience and support, which made it happen. I want to thank Prof. Maninder Singh, Head, Department of Computer Science & Engineering, for his kind co-operation. Many thanks go to the research committee members, namely, Dr. Inderveer Chana, Dr. Neeraj Kumar and Dr. M.D.Singh.

Words fail to express my deep gratitude to Dr. Manoj Sharma for his guidance, motivation and support throughout this journey. Many thanks to Dr. Rajesh Kumar, Dr. Anju Sharma, Mr. Jitender Kumar and Ms. Monika Sethi for their help during the course work. Besides, I am grateful to Mr. S.P.Sachdeva, Chairman, Sachdeva Groups of Educational Institutions, for his kind-co-operation throughout my doctoral study. Thanks to my institutional heads, namely, Dr. S.S.Bedi, Dr. G.S.Virdi and Brig. (Dr.) R.S.Grewal, for their co-operation.

I take this opportunity to remember and express my deep gratitude to my teachers, friends, and colleagues, who have done exemplary work in their profession, and are a constant source of inspiration to me. Last but not least, I am indebted to my family members for their unending support.



(Deepak Gupta)

Abstract

The world is already into the information age. The huge growth of digital data has overwhelmed the traditional systems and approaches. Big data is touching almost all aspects of our life and the data-driven discovery approach is an emerging paradigm for computing. The ever-growing data provides a tidal wave of opportunities and challenges in terms of data capture, storage, manipulation, management, analysis, knowledge extraction, security, privacy, and visualization. Though the promise of big data seems to be genuine, still a wide gap exists between its potential and realization.

In this era of digitization, a huge amount of data being generated has resulted in an exponential growth of widespread cyber threats. Moreover, the ever-evolving threat landscape and rapidly growing network environments are offering additional ways for the attackers to break in. This scenario has overwhelmed the existing traditional solutions and rendered them outdated to handle such attacks. To encounter the real-world cybersecurity challenges, the security researchers are putting a lot of efforts on technologies stemming from areas like big data, and artificial intelligence to extract powerful insights. Malware is one of the most critical and challenging security threats in the Internet world. It is growing exponentially in terms of volume, variety and velocity, and thus overwhelms the traditional approaches employed for malware detection and classification. Moreover, with the advent of Internet of Things, there is a huge growth in the volume of digital devices and in such scenario, malicious binaries are bound to grow even faster making it a big data problem.

The main aim of this research is to explore the various tools and techniques of big data processing and analysis, and propose a framework for analyzing big data to generate the actionable insights or intelligence. A case study of malware analysis and detection has been used in the research. The initial part of the research focuses on understanding the basic concept of big data, its evolution and popular open-source big data stream processing frameworks. A bibliometric study of academic and industry publications during the period 2000–2017 is conducted to understand the current state, evolving

disciplines, tools and techniques, and research trends of big data. A comparison of the most widely used open source big data stream processing frameworks is made, and the major big data research challenges and directions are identified and deliberated to offer rich observations and thinking. Further, the significance of big data analytics and machine learning in cyber security is identified.

An enormous amount of malware samples is available online, but only a few researchers have attempted to analyze these thoroughly for obtaining insights or threat intelligence by extracting and analyzing behavioral trends using big data frameworks. This type of trend analysis could be very useful to understand the context and the goals of security breaches. In this research, we have proposed a scalable architecture built on the top of Apache Spark to perform a statistical analysis to study malware behavioral trends during the period 2010 to 2017. These trends can be further extrapolated by security experts to generate cyber threat intelligence which can help organizations to improve their threat protection systems and reduce the risks posed by malicious binaries.

In order to analyze and detect unknown malware on a large scale, security analysts need to make use of machine learning algorithms along with big data technologies. These technologies help them to deal with current threat landscape consisting of complex and large flux of malicious binaries. This research proposes the design of a scalable architecture using Apache Spark and its scalable machine learning library for detecting zero-day malware. Three machine learning algorithms, namely, Naïve Bayes, support vector machine and random forest are used and the experimental results show that random forest gives the best accuracy.

Although, many machine learning models have been used in detection and classification of malicious binaries in literature, however, the performance of ensemble learning methods has not been investigated extensively on large malware data. We have designed two methods based on ensemble learning and big data for improving the performance of malware detection at large scale. The first method is based on the weighted voting strategy of ensemble learning, and the second method is for selecting an optimal set of base classifiers for stacking purpose. The proposed methods are implemented using Apache Spark, and their performance is tested and evaluated. The experiments demonstrate that the proposed approach improves the generalization performance in detecting new malware as compared to traditional ensemble methods.

Increasing complexity and sophistication of malware has led to many state-of-art machine learning based solutions. However, many of these solutions suffer from high false positive rates and low scalability restricting their wider adoption and deployment. In recent years, deep learning, a subfield of machine learning, has resurged and reported outstanding performance in tackling many classification problems in a wide range of fields. A deep learning model has been proposed for malware detection which uses Apache Spark for efficient data preprocessing and Keras with TensorFlow as its computational engine for implementation of deep learning model. The findings demonstrate that the four layer deep learning model achieves the highest accuracy.

The present research provides an evidence-based knowledge pertaining to application of big data tools in malware detection. It provides a comprehensive study of big data evolution including batch and stream processing tools, bibliometric analysis, and research challenges. It includes the study of big data security analytics which identifies malware detection and classification as a big data problem. Finally, the research proposes a set of scalable solutions for malware detection at large scale. These solutions are developed on the top of Apache Spark and Keras along with TensorFlow, and use machine learning, ensemble learning, and deep learning techniques to identify the malicious binaries.

Chapter 1: Introduction

The world is already into the information age. The huge growth of digital data has overwhelmed the traditional systems and approaches. Big data is touching almost all aspects of our life and the data-driven discovery approach is an emerging paradigm for computing. The ever-growing data provides a tidal wave of opportunities and challenges in terms of data capture, storage, manipulation, management, analysis, knowledge extraction, security, privacy, and visualization. Though the promise of big data seems to be genuine, still a wide gap exists between its potential and realization. In last few years, there is a huge surge in research efforts in academia as well as industry to have a better understanding of big data. This chapter provides a basic concept of big data, its evolution including a bibliometric study of academic and industry publications during the period 2000–2017, and popular open-source big data stream processing frameworks. The chapter also discusses prevalent research challenges which must be addressed to realise the true potential of big data.

1.1 Big Data Deluge

The digital data seems to be there everywhere. The data deluge is happening at the unmeasurable rate and is growing continuously. Hundreds of petabytes of digital data is being continuously generated from sources like social media, mobile devices, sensors, individual archives, astronomy, cameras, software logs, scientific instruments, financial market data, health data etc. Most of the data is locked up in the largest web properties (like search engines, social media), Internet of Things (IoT), scientific and financial institutions. In fact, we create a digital trail nearly every minute. As of March 2018, Facebook has 2.2 billion monthly and 1.45 billion daily active users, which is around 13% more than the previous year. There are 1.5 billion monthly active WhatsApp users [1]. Around 500 million tweets are sent on Twitter every day, which means that there is a 10-

fold increase in the last six years. On a daily basis, more than 7 billion searches are being made on Google. Moreover, the digital universe has already seen the impact of IoT [2]. In 2013, the data from embedded systems (like sensors, systems monitoring physical universe) was 2% of the size of the digital world and is expected to grow to 10% by 2020 [3]. In astronomy, the Australian Square Kilometre Array Pathfinder (ASKAP) presently obtains 7.5 terabytes (TB) of data per second and is expected to become 750 TB per second (approximately 25 zettabytes (ZB) per year) by 2025. There is an estimate of sequencing human genomes between 100 million and 2 billion by 2025, and 2-40 exabytes (EB) of space would be needed for storing these genomes [4]. The Large Hadron Collider (LHC) may store data at 10 gigabytes (GB) per second in future runs and the data is analysed by computing grid consisting of more than 170 computing centres in 36 countries [5]. These staggering statistical figures depict a tremendous growth of data in various disciplines. In 2013, the total size of the digital universe was 4.4 ZB and is expected to grow 40% per year into next decade, nearly doubling its size after every two years. The digital universe is likely to reach 44 ZB, or 44 trillion GB by 2020. In 2013, mature markets like Germany, Japan, and the US generated 60% of the size of the digital universe. By 2020, the emerging markets like Brazil, China, India, Mexico, and Russia will generate most of the data [3].

Big data is a term used to represent the deluge of data in information-driven world and many in academia are seeing it as a beginning of the fourth paradigm of data-driven scientific research [3], shortened to data science. Data science is another popular term related to prediction, data manipulation, visualization and other similar tasks. With the increase in the scale of data, data science helps to extract novel insights from data which were not feasible earlier with small amounts of data. Apart from data science, the commonly used terms to describe data analysis are knowledge extraction, knowledge or information discovery, information harvesting, data archaeology, data pattern processing and exploratory data analysis [6]. Big data has the potential to make an impact on domains like science, healthcare, government, defence, education, economics, humanities and almost every other aspect of society as well. In general, big data is affecting most of the aspects of our life and the data-driven discovery approach is an emerging paradigm for computing.

The plethora of data provides equal opportunities and challenges. Making sense of it is the key point. Any organization, which successfully harness the power of this huge tidal

wave of information will jump ahead of its competitors. The top industry organizations like Google, Microsoft, Facebook, Twitter, Amazon, IBM, Intel etc. are deeply engaged in big data research and its services. There are numerous big data based organizations like Cloudera, Databricks, Greenplum, Hortonworks, MapR, Splunk, Tableau, Teradata, VMware etc. Top scientific journals and magazines like Nature and Science, started discussions on big data opportunities and challenges through special issues [7, 8]. Thereafter, many academic journals and magazines have incorporated special issues on big data leading to a huge surge in research publications in recent years. The earlier issues focussed on the basic understanding of big data. As the concepts and technologies are maturing and evolving, the focus is to make intelligent predictions or deeper insights using machine learning, deep learning, and artificial intelligence techniques. Many top institutions like MIT, UC Berkeley, University of Cambridge, etc. have established their own big data research and innovation centres. Special courses on big data are being offered by numerous institutions and are in huge demand across the world. The public media has also made significant contributions to spread out awareness about the developments in this exciting field [9, 10]. Recognising the value of big data, many government agencies have also started projects related to big data research and applications [11, 12]. Global Pulse, a popular big data initiative of the United Nations, aims to harness big data for sustainable development and is involved in projects related to climate & resilience, data privacy & protection, food & agriculture, public health, humanitarian action, gender and real-time evaluation of data [13].

The fundamental challenge for the big data is to analyze the data on a large scale and extract extensive information or knowledge for future actions, which has become more difficult than ever. It is necessary to understand that the value of big data lies in finding insights, still it is more important to take actions on the basis of these insights. The problem is that the existing conventional tools are not capable of processing and analysing the data on such a large scale. Moreover, the data is being generated on a regular basis in most of big data applications and such streaming data requires even more advanced tools to process it in near real-time. Overall, the rate of growth of big data is outpacing the technical advances in data storage, analysis, management and more, and thus, posing enormous challenges to the researchers. The interesting thing is that this problem is not domain specific.

1.2 Big Data Evolution

The big data is evolving very fast and the huge influx of data is only going to increase in future. This section provides a brief history, popular definitions and key characteristics of big data followed by its classification.

1.2.1 Brief History

The origin of the term big data, in its present connotation, goes to John Mashey, a chief scientist at Silicon Graphics, who delivered many talks to explain the concept in the late 1990s. In 1998, he presented a slide entitled – 'Big Data and the Next Wave of InfraStress'. In 1998, the first academic reference in computer science was made by Weiss and Indurkhya in his book – 'Predictive Data Mining: A Practical Guide'. In 2000, the first academic reference in statistics was made by Francis X. Diebold in his paper – 'Big Data Dynamic Factor Models for Macroeconomic Measurement and Forecasting' [14]. In 2001, Douglas Laney, an analyst with the Meta Group (now Gartner), published a research article – '3D Data Management: Controlling Data Volume, Velocity, and Variety' [15] and thus, enriched the concept by introducing 3Vs of big data - volume, velocity and variety.

1.2.2 Defining Big Data

Despite the widespread consensus on the promise and opportunities of big data, there is no standard definition of it at present, though efforts are on to find out a precise definition [16, 17]. There have been dubious statements like size is the only thing which matters in big data, and for others, it may be small dataset which is frequently changing. In fact, big data is a relative term as today's big data may be tomorrow's small data. A few widely accepted definitions of big data are given in Table 1.1. We propose a definition of big data as follows:

Big data refers to large datasets which require non-traditional scalable solutions for data acquisition, storage, management, analysis, and visualization; aiming to extract actionable insights with a potential to impact every aspect of human life.

Table 1.1: Popular big data definitions.

Author	Year	Definition	Reference
McKinsey Global Institute (MGI)	2011	"Big data refers to a dataset whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze."	[18]
Gartner	2012	"Big Data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."	[19]
V. Mayer-Schonberger, and K. Cukier	2013	"Big data refers to things one can do at a large scale that cannot be done at a smaller one, to extract new insights or create new forms of value, in ways that change markets, organizations, the relationship between citizens and governments, and more."	[20]
National Institute of Standards and Technology (NIST)	2015	"Big data consists of extensive datasets - primarily in the characteristics of volume, variety, velocity, and/or variability - that require a scalable architecture for efficient storage, manipulation, and analysis."	[16]
International Data Corporation (IDC)	2017	"Big data software is described as a new generation of software and architectures designed to economically extract value from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis."	[21]

1.2.3 Key Characteristics

Big data can be described more precisely by a set of following distinguished characteristics:

- **Volume:** The volume refers to the size of the dataset that organizations are trying to harness to improve decision-making. The data volume is continuously increasing at an unprecedented rate and thus, it is not uncommon to have a sizeable number of organizations generating petabytes of data. The data may come from click streams, transaction histories, sensors and so on. There is no well-defined quantity of data to be known as big data. In general, big data size ranges from a few terabytes to petabytes or exabytes. The rate of growth of data will keep on increasing in future and hence, this range will retain its upward journey.

- **Velocity:** Velocity is the most misunderstood aspect of big data and is often equated with real-time analytics. Velocity means the rate at which data is generated, processed, analysed and visualized. Analysis of streaming data helps to make the decision within fractions of a second. There are two main reasons to do streaming processing – first, to keep storage requirements at a practical level for big data and second, when the application mandates immediate response to the data, which is a typical situation in financial, mobile and online gaming applications.

- **Variety:** Variety is one of the most exciting aspects of big data. It represents the heterogeneity of data types. The data is not just numbers; data can be words, images, tweets, videos, logs or any other content. Around 80% of the data in this universe is estimated to be semi-structured or unstructured and thus, cannot directly integrate into an application. Moreover, it is essential to tag data to know more about it - a practice that results in the metadata. Metadata is one of the fastest-growing sub-segments of the digital universe and it provides additional information about the data itself. In case of a video file, metadata refers to the date, time and location of a video file.

- **Variability:** Variability means the change in data like flow rate, format or composition, over a period of time. In such scenarios, the data processing requires automatic provisioning of resources in cloud computing environment. The automated provisioning allows addition or removal of the nodes, depending on availability of data, and provides a huge cost saving in big data processing.

- **Veracity:** Veracity describes the completeness, accuracy and quality of data. It is related to reliability or consistency of some data types. In case of causal analytics, every data element is critical. In corrections based analytics, it is otherwise. A few examples of uncertain data are sentiment and truthfulness in humans, weather forecasting and economic factors. Veracity allows big data to consider such uncertainty in data and this can be managed by tagging data properly.

- **Value:** This is what the organizations are looking for, i.e., business value. It describes the rich benefits reaped by the organizations by drawing insights from the data and transform these insights into actions which were not feasible earlier.

- **Validity:** Validity means correctness of data for its intended use. Data cleaning is the most cumbersome, time-consuming and critical task before doing any data analytics. The benefits of data analytics depend upon the correctness of data.

1.2.4 Big Data Classification

Table 1.2 shows big data classification which intends to provide a view of various elements of big data at a broad level. The classification represents big data sources, characteristics, format, storage types, analytics phases and types, open source streaming tools and techniques, major research challenges and opportunities in interdisciplinary domains, use cases, and ultimate goals of big data. The various categories are briefly described below:

- Big data is generated from heterogeneous sources and can be described by using several characteristics. These diverse sources can produce a vast amount of raw data comprising of around 80% of unstructured or semi-structured data, which can be stored in different formats.

- Big data analytics (BDA) is a multi-step process, starting from data acquisition and ending with visualization. For efficient large-scale analysis, these steps should be automated which demands automatic resolution of various data structures and semantics. Besides, there are four types of analytics. The predictive and prescriptive analytics belong to advanced analytics, which uses comparatively complex mathematical modelling including machine learning, natural language processing, statistics, operations research, etc.

- New big data tools and techniques are required to realize the real potential of big data. The advances and innovations in various big data tools and techniques will provide better knowledge discovery from data. Apache Hadoop has already become the de facto standard for big data implementations. In addition, Apache Hadoop ecosystem contains many big data processing tools and the additional projects surrounding Hadoop. Presently, Apache Spark is the most popular stream processing framework. The stream processing tools help to keep the storage at a practical level as well as provide immediate response to the data. In recent times, the innovations in machine learning are staggering. With the proliferation of big data, machine learning has become the forefront of research and industry applications. At the same time, machine

Table 1.2: Big data classification.

Big Data Classification					
Sources	**Characteristics**	**Storage Types**	**Analytics Types**	**Techniques**
Social Media	Volume	Wide-Column Store	Descriptive	Machine Learning
Internet of Things	Velocity	Document Store	Diagnostic	Data Mining
Web 2.0 Data	Variety	Key-Value Store	Predictive	Statistics
Transactional Data	Variability	Graph Store	Prescriptive	Artificial Intelligence
Software Logs	Veracity			Cognitive Computing
Enterprise Applications	Value	**Analytics Phases**	**Open Source Tools**	Deep Learning
Science and Research	Validity	Acquisition	Apache Hadoop	Crowdsourcing
Healthcare		Extraction	Apache Hadoop Ecosystem	Optimization
Astronomy	**Format**	Cleaning		Natural Language Processing (NLP)
Genomics	Structured	Filtering	Apache Spark	
Economic Data	Semi-structured	Validation	Apache Storm	Signal Processing
Retail & Manufacturing	Unstructured	Integration	Apache Samza	Optimization Methods
Telecommunication		Analysis	Apache Flink	
Public Sector		Visualization	Apache Apex	Visualization

Research Challenges	Target Domains	Use Cases	Ultimate Goals
Data Acquisition and Metadata	Finance and Insurance	Sentiment Analysis	New Innovations in all Disciplines
Data Quality	Healthcare	Social Network Analysis	Decent Economic Growth
Data Storage	Retail & Manufacturing	Clickstream Analysis	Good Health
Data Sharing and Transfer	Social Media	Real-time Analytics	Better Quality of Life
Scalability	Media and Entertainment	Log Analytics	Quality Education
Data Analysis	Gaming	Fraud Detection	Peace and Justice for all
Data Query and Indexing	Banking	Recommendation System	Gender Equality
Data Uncertainty	Cyber Security	Credit Scoring	Decent Work for all
Data Privacy, Security and Ethics	Transportation	Smart Pricing	Eradicate Hunger & Poverty
Data Visualization	Education	Stock Market Prediction	Clean Water & Sanitation
	Energy	Operational Analytics	Clean & Affordable Energy
	Telecommunication	Image Processing	Resilience to Natural Disasters
	Science & Research	Biomedicine	Ecosystem Conservation
	Astronomy	Smart Homes and Cities	Sustainable Transport
	Social Development	Energy Distribution System	Transparent, Accountable and Effective Governance
	Crime and Security	Cyber Threat Intelligence	
	Natural Disasters Management	Climate Change	
		Regulatory Compliances	

learning has never been challenged before as the traditional machine learning algorithms face many critical challenges to transform big data into actionable intelligence. Very recently, there is an exceptional growth in deep learning. The deep learning models have the ability to learn though multiple layers for making decisions rather than relying on domain experts for feature selection.

- Big data research has numerous prevailing and emerging challenges. It adopts an interdisciplinary approach to solve real-life use cases and the ultimate goal is to change the way we live.

1.3 A Bibliometric Study of Research Publications in Academia and Industry

To better understand the current state and research trends of big data, a bibliometric study of relevant literature in academics and industry has been conducted. It is performed by collecting, filtering, storing and analysing the literature information over a span of 18 years (2000 - 2017) targeting following five mainstream digital libraries:

a) ACM Digital Library, which offers access to ACM publications, including journals, conferences and technical magazines.
b) IEEE Xplore provides access to scientific and technical contents published by the IEEE.
c) SAGE Journals (world's 5th largest journals publisher, having more than 1000 journals covering multiple domains).
d) ScienceDirect offers articles from over 2,500 journals in engineering, science (physical, life and health) and humanities.
e) Web of Science (the largest citation index offers access to 100+ years of abstract).

1.3.1 Methodology

In each of the above mentioned libraries, a keyword “big data” is searched in the topic (either title or abstract or keywords) of a research publication to analyse the literature published between 2000 and 2017. Only academic journals, magazines, and conferences articles are considered. Starting with ACM Digital Library, a search query is executed on ‘The ACM Full-Text Collection’ for individual years, whose syntax for 2017 looks like:

```
"query": { acmdlTitle:(+"big data")
        OR recordAbstract:(+"big data")
        OR keywords.author.keyword:(+"big data") }
"filter": {"publicationYear":{ "gte":2017, "lte":2017 }}, {owners.owner=HOSTED}
```

In IEEE Xplore, the search is conducted for individual years by selecting the content types - conference publications, journals & magazines, early access articles, and standards. In SAGE Journals, an advance search choosing all contents is done separately for title, abstract and keywords as there is no provision to search for these in one go. In ScienceDirect, an advanced search is made for individual years by selecting article, review article, short survey, short communication, and editorial under 'Journals' tab. In Web of Science, a basic search is performed for every year on the core collection of publications, namely Science Citation Index Expanded (SCI-EXPANDED), Social Sciences Citation Index (SSCI), Arts & Humanities Citation Index (A&HCI), and Emerging Sources Citation Index (ESCI).

The high-quality bibliometric metadata consisting of author names, article title, publication title, keywords, and year of publication, is stored in MS Access 2013 database. At the outset, the number of records stored are 28,411. In the next step, the MS Access Query Wizard is used to locate and remove duplicate articles from every table. Further, the proceeding instructions/reports, blogs, authors' profiles, editors' messages/letters, tutorials, lectures, discussions, conferences or workshops summaries/reports, program instructions, posters, presentations, keynote talks/address, interviews, seminars, special issue notes, welcome notes, and panel sessions are also removed. As of April 2018, the number of unique matching research articles are found to be 24,662. The various steps involved in bibliometric data analysis are shown in Figure 1.1.

1.3.2 Key Findings

This subsection discusses the key findings of the bibliometric study pertaining to big data and is further divided into two subtopics: top journals and magazines, and title words and author-assigned keywords statistics. The former subtopic helps researchers to identify the leading journals and magazines in publications to strengthen their work, and the latter provides valuable information about the research trends.

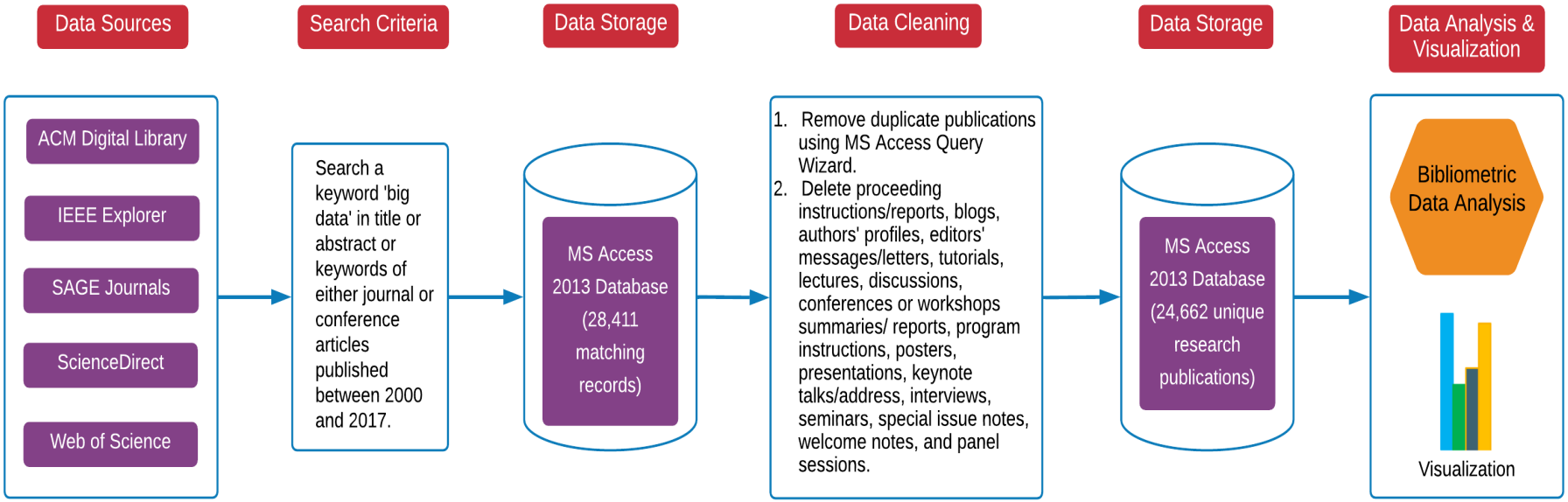
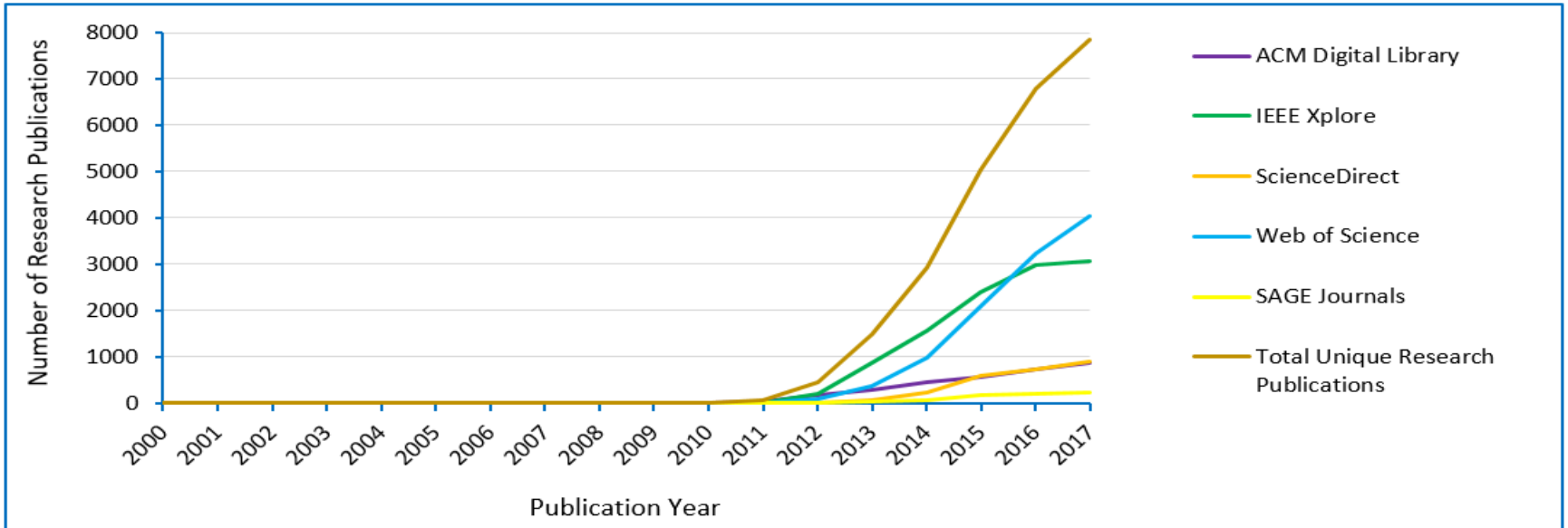


Figure 1.1: The data analysis pipeline pertaining to bibliometric study of research publications.



Digital Library	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
ACM Digital Library	0	1	1	0	0	1	0	0	0	3	8	28	188	280	461	578	740	858
IEEE Xplore	0	2	1	0	0	1	1	2	3	6	5	15	193	878	1576	2395	2997	3068
ScienceDirect	0	0	0	0	0	0	1	1	1	1	0	3	12	73	220	592	725	887
Web of Science	0	0	1	2	2	0	3	1	11	6	5	22	92	384	991	2087	3236	4041
SAGE Journals	0	0	0	0	0	0	0	0	0	0	0	0	8	41	63	188	194	227
Total Records	0	3	3	2	2	2	5	4	15	16	18	68	493	1656	3311	5840	7892	9081
Total Unique Research Publications	0	2	3	2	2	2	4	4	15	14	18	62	452	1486	2926	5039	6795	7836

Figure 1.2: Distribution of research publications pertaining to big data published from 2000 to 2017.

Figure 1.2 depicts growth trends of research publications pertaining to big data. A closer look at the bibliographic study shows that the research interest in this interdisciplinary field has got much attention in recent times, especially after 2010-2011 and is still growing. Only 62 unique research articles are indexed in 2011. This number has ascended to nearly 7,836 in 2017. This means that 126 times more papers are indexed in 2017 as compared to 2011. Overall, 99.5% of the total research articles are published in last six years, i.e. 2012-2017. Looking at the distribution of publications by different digital libraries, IEEE Xplore (11,143) has the highest number of big data publications followed closely by Web of Science (10,884). The steep rise in the publications indicates the growing popularity and relevance of big data research, which is gradually evolving and maturing with time. It is likely that the increasing trend of journal publications will remain intact in the following year 2018.

1.3.3 Top Journals and Magazines

The knowledge of popular publications is an essential aspect of any bibliometric analysis as it provides insights about evolving research disciplines. In general, the academic conferences are the largest source of publications. The research articles published in conferences (12,571) are closely followed by journals and magazines articles (12,091). Table 1.3 lists the top 10 journals and magazines with big data publications. These journals and magazines are most active in publishing research on big data. IEEE Access, a multidisciplinary open access journal, has published maximum research articles about big data. This is closely followed by Big Data & Society, another open access peer-reviewed interdisciplinary journal for publishing research about big data practices. In magazines, Computer - a flagship magazine of the IEEE Computer Society, has the maximum number of published articles. Moreover, the top seven magazines in the list belong to the IEEE publications.

1.3.4 Title Words and Author-Assigned Keywords Statistics

Another analysis is performed to identify the commonly used words in publication titles and the list of keywords. These can be used to understand the research trends and future tracks for big data research. Table 1.4 and 1.5 provide a list of top 30 frequently used words in paper titles and author-assigned keywords respectively. These lists are obtained from a total of 24,662 research publication titles and 1,42,702 keywords.

Table 1.3: Top 10 journals and magazines with big data publications.

Rank	Journal	Publications	Magazines	Publications
1.	IEEE Access	134	Computer	89
2.	Big Data & Society	130	IT Professional	58
3.	International Journal of Distributed Sensor Networks	110	IEEE Internet Computing	43
4.	IEEE Transactions on Big Data	107	IEEE Intelligent Systems	38
5.	Future Generation Computer Systems	106	IEEE Cloud Computing	37
6.	PLOS ONE	102	IEEE MultiMedia	35
7.	Big Data	83	IEEE Communications Magazine	30
8.	Concurrency and Computation: Practice and Experience	81	Communications of the ACM	29
9.	Cluster Computing - The Journal of Networks, Software Tools and Applications	77	XRDS: Crossroads, The ACM Magazine for Students	26
10.	Neurocomputing	72	Science	24

A closer look at the tables shows that many title words and keywords are interrelated, which seems reasonable as both of these are provided by the author. Further, the data distinctly depicts that the keywords like big data, data analytics, cloud computing, MapReduce, Hadoop, data mining, systems, machine learning, model, and algorithm have been consistently used by the researchers in the literature. In addition, the high-frequency of some words shows that the researchers as well as organizations are developing and deploying new technologies (e.g., cloud computing, distributed computing, database, security, Hadoop, MapReduce, and framework) and techniques (e.g., classification, clustering, data mining, machine learning, optimization, model, learning, algorithm, social media/network, management, and visualization) to harness the value of big data. As the big data techniques and technologies are being developed all the time and thus, the researchers have to address considerable challenges to realize the real value of big data. In the next Section 4, we will discuss the popular open source big data streaming technologies followed by the major research challenges in Section 5.

Table 1.4: Top 30 frequently used words in the title of research publications pertaining to big data.

Rank	Word	Count	Rank	Word	Count	Rank	Word	Count
1.	Data	14451	11.	Computing	931	21.	Algorithm	755
2.	Big	9957	12.	Framework	930	22.	Smart	741
3.	Analysis	2096	13.	Social	926	23.	Information	735
4.	Analytics	1773	14.	Large	914	24.	Performance	725
5.	Cloud	1490	15.	Management	899	25.	Scale	715
6.	System	1288	16.	Network	888	26.	Time	709
7.	Learning	1131	17.	Model	846	27.	Networks	705
8.	Research	1106	18.	Mining	803	28.	Efficient	673
9.	Approach	1022	19.	Distributed	794	29.	Challenges	668
10.	Processing	939	20.	Applications	772	30.	MapReduce	667

Table 1.5: Top 30 commonly used author-assigned keywords.

Rank	Keyword	Count	Rank	Keyword	Count	Rank	Keyword	Count
1.	Big data	7984	11.	Network	659	21.	Performance	442
2.	Data Analytics	1739	12.	Internet of Things	658	22.	Data	430
3.	Cloud Computing	1495	13.	Social Media	646	23.	Security	395
4.	MapReduce	1244	14.	Distributed Computing	557	24.	Information	388
5.	Hadoop	923	15.	Healthcare	545	25.	Challenges	371
6.	Data Mining	883	16.	Privacy	538	26.	Social Network	353
7.	Systems	792	17.	Classification	524	27.	Framework	324
8.	Machine Learning	790	18.	Visualization	506	28.	Database	305
9.	Model	748	19.	Management	495	29.	Technologies	302
10.	Algorithm	671	20.	Clustering	468	30.	Optimization	292

1.4 Open Source Big Data Stream Processing Frameworks

Big data requires non-conventional tools to store, manage, process and analyze data within a tolerable elapsed times. There is a lot of innovation and development happening about big data tools in free and open-source software system. Earlier, Apache Hadoop has become the de facto standard for big data implementations. The promise of Hadoop largely remains unfulfilled because of the complexity associated with MapReduce and the need to process data in motion in particular. Thus, it is necessary to move beyond Hadoop MapReduce processing model. In fact, the big data stream processing frameworks are gradually gaining popularity as most of the big data applications are generating data on a continual basis, and IoT era will further strengthen adoption of such technology in future [22]. Presently, Apache Spark [23], a fairly new stream processing platform, is preferred due to ease-of-use and relatively better performance than Apache Hadoop. In this section, we first discuss Apache Hadoop - a batch processing framework, followed by the most widely used open source big data stream processing frameworks. At last, we provide the comparison of stream processing frameworks (in Table 1.7).

1.4.1 Apache Hadoop

Apache Hadoop is a framework for the batch processing of data at a large scale and storage in a distributed environment using commodity hardware. Hadoop was created in 2005 and was implemented by using Java language. Hadoop does processing of data across a cluster of machines using the underlying parallelism of the CPU cores. Hadoop core components include Hadoop Distributed File System (HDFS) [24], Hadoop MapReduce (a data processing engine) [25] and Hadoop YARN (Yet Another Resource Negotiator) for better job scheduling and resource management [26]. The main benefits of Hadoop are high scalability, reliability, performance and low cost. Hadoop also offers a trustworthy shared storage and data exploration system. HDFS is developed on the lines of Google File System (GFS) [27], which splits the large data files into smaller chunks distributed over different nodes of a cluster. Each chunk is replicated across several machines making it fault-tolerant. The MapReduce framework has got a lot of attention for data-intensive applications [28]. It hides the details of parallel execution [29]. The key advantages of MapReduce include fault tolerance, efficient data processing and better execution [30, 31]. YARN separates resource management and programming framework to provide more scalability, better efficiency, and sharing of a cluster among various frameworks.

1.4.2 Apache Spark

Apache Spark is a highly reliable and scalable processing engine, scaling from a single server to thousands of machines. It introduces the concept of a resilient distributed dataset (RDD) [23], which is a collection of fault-tolerant elements that can be operated in parallel, improving the performance of iterative distributed computing. Spark does in-memory processing and runs much faster than Hadoop MapReduce [32]. As compared to Apache Hadoop MapReduce, it performs around 100 times better in memory and 10 times better on disk. The sort benchmarks [33] observed the processing time 23.4 minutes on Apache Spark (using 207 nodes) as compared to 72 minutes on Apache Hadoop (using 2100 nodes) to process 100 TB (on disk) of data. Table 1.6 makes a comparison between Apache Hadoop and Apache Spark.

Apache Spark contains libraries like Spark Streaming for fault-tolerant stream applications, MLlib for scalable machine learning, Spark SQL for working with structured data, and GraphX for graph computation. It can access data stored in HDFS or any storage systems supported by Hadoop APIs like local filesystem, Amazon S3, Cassandra, Hive, HBase, etc. Apache Spark can be deployed in two ways - stand-alone, and distributed computing mode.

1.4.3 Apache Storm

Apache Storm [34] is a pioneer in large scale streaming data processing system. In 2010, it was developed by Nathan Marz and his team at BackType. Later, Twitter acquired it and in 2013, it was incorporated in Apache incubator program. It is quite easy to use and can be used with any programming language. Apache Storm cluster consists of three type of nodes: Nimbus (uploads data, launch workers, monitor computations), ZooKeeper (for cluster management) and Supervisor (starts and stops workers). The main benefits of Apache Storm include high speed, scalable, fault-tolerant, reliable and easy to operate platform. The use cases include real-time analytics, data monetization, cybersecurity analytics and threat detection.

1.4.4 Apache Samza

Apache Samza [35] is a near real-time distributed data processing framework. It was developed along with Apache Kafka, a low latency messaging system. Before becoming a part of Apache incubator, both Samza and Kafka were developed by LinkedIn. Samza

performs computations continuously as and when the data arrives resulting in a very low response time. There are three layers in Samza architecture - a streaming layer (Apache Kafka), an execution layer (YARN) and a processing layer (Samza API). There is an inbuilt support for Apache Kafka and YARN. The key benefits of Apache Samza include scalability, fault tolerance, managed state, pluggable API, resource management, security and processor isolation.

Table 1.6: Comparison between Apache Hadoop and Apache Spark platforms.

Comparison Metrics	**Apache Hadoop**	**Apache Spark**
Created in	2005	2009
Induction into Apache Incubator Program	October, 2007	June, 2013
Most Stable Release	3.1.2	2.4.3
Implemented in	Java	Scala
Programming APIs	Java	Java, Scala, Python, R
Programming Model	Key/Value pair	Function
Storage	Disk	Memory
Data Processing	Batch	Batch and streaming
Data Caching	No	Yes
Iterative Processing	No	Yes
Machine Learning Library	Apache Mahout	Inbuilt library (MLlib)
Fault Tolerance	Very High	High
Latency	High	Medium (seconds)
Speed	Less	100 times faster in memory, 10 times faster on disk faster
Processing Speed to Process 100 TB (on Disk) (www.sortbenchmark.org)	72 minutes (2100 nodes)	23.4 minutes (207 nodes)
Deployment Mode	Standalone, cloud, hybrid	Standalone, cloud, hybrid
Ease-of-Use	Complex	Easy
Project Cost	Low	High
Project Maturity	Very High	High

Table 1.7: Comparison of open source big data stream processing frameworks.

Comparison Metrics	**Apache Spark**	**Apache Storm**	**Apache Samza**	**Apache Flink**	**Apache Apex**
Created in	2009	2010	Around 2010/2011	2011	2012
Induction into Apache Incubator Program	June, 2013	Sept, 2013	July, 2013	April, 2014	August, 2015
Most Stable Release (as of April 30, 2018)	2.3.0	1.2.1	0.14.0	1.4.2	3.7.0
Implemented in	Scala	Scala, Clojure	Java, Scala	Java, Scala	Java, Scala
Language Support	Java, Scala, Python, R	Java, Clojure, Scala, Python, Ruby, any other language.	Java, Scala JVM-languages only	Java, Scala, Python, SQL	Java, Scala
Data Processing Model	Inside memory	Tuple	Message	Inside memory	Inside memory
Computation Model	Micro-batch	Native	Native	Native	Hybrid
API Model	Declarative	Compositional	Compositional	Declarative	Compositional
Guarantees	Exactly once	At least once	At least once	Exactly once	Exactly once
Machine Learning Library	Inbuilt library (MLlib)	Apache Scalable Advanced Massive Online Analysis (SAMOA)	Apache SAMOA	Inbuilt library (Flink ML), Apache SAMOA	Apache SAMOA
Deployment/Resource Management	Standalone, YARN, Mesos	YARN, Mesos	YARN, Mesos,	Standalone , YARN, Mesos	YARN

Development Cost	Low (same code for batch and stream processing)	Moderate (different code for batch and stream processing)	Reasonable	Reasonable	Reasonable
State Management	Stateful	Stateless	Stateful	Stateful	Stateful
Event Processing Guarantee	Exactly once	At least once	At least once	Exactly once	Exactly once
Fault Tolerance	Resilient distributed dataset (RDD) based checkpointing	Record acknowledgements	Log based	Checkpointing	Checkpointing
Latency	Medium (seconds)	Very low (sub-second)	Low (sub-second)	Low (sub-second)	Very low (sub-seconds)
Throughput	High	Low	High	High	High
Speed	100x (in memory), 10x (on disk) faster than MapReduce	Streaming is 40x slower than Spark	High	Better than Spark	High
Project Maturity	High	High	Medium	Medium	Medium

1.4.5 Apache Flink

Apache Flink [36] is an open-source run-time data processing framework, written in Java and Scala. Flink was developed at Berlin's Technical University. It was known as Stratosphere before its inclusion in Apache incubator program. It is designed to run on clusters having thousands of nodes. At the basic level, a Flink program consists of a data source, data transformations and a data sink to which it sends data after processing. It provides a replacement for Hadoop MapReduce using directed graph approach which uses in-memory processing giving a far better performance. The main benefits of using Apache Flink are high-throughput and low latency, fault tolerance, flexible windowing and version maintenance (enables easy reprocessing of historical data).

1.4.6 Apache Apex

Apache Apex [37] framework processes big data-in-motion natively in Hadoop. It was named top-level project by Apache Software Foundation in 2016. Apache Apex has two parts: Apex Core and Apex Malhar. Apex Core is a framework for developing applications on Apache Hadoop and is supplemented by a library, Apex Malhar, of pre-built operators. Apache Apex is highly scalable, high throughput, fault-tolerant, auto-memory management, easily operable and very low latency platform.

Performance benchmarks [38-41] for real-time processing frameworks suggest that the competition is catching up among such frameworks and at present, there is no clear winner, though some benchmarks identify strengths of a particular framework. The choice of a particular framework should be carefully made on the basis of use cases as every framework has its own advantages and disadvantages.

We have used Google Trends [42] service to comprehend Google users' search interest in aforementioned big data processing frameworks to draw better insights and comparisons. The searches for different frameworks are performed on a large static dataset by selecting the options - ‘Woldwide’, ‘All categories’, and ‘Web Search’, over a period between 2010 and 2017. Since Google Trends can compare at most five search terms at a time, a couple of comparisons are made to include all six search terms. The first comparison included five search terms - Apache Hadoop, Apache Spark, Apache Storm, Apache Samza and Apache Flink. The second comparison included only two search terms – Apache Hadoop and Apache Apex. The data of each comparison is exported to a comma-separated values (CSV) file and after normalization, merged into a single file having all six search terms

to eventually understand and compare search interest for big data processing frameworks over a period of 8 years, starting from 2010 to 2017. The interest over time represents the search interest w.r.t highest point on the graph for a given time. The maximum search interest for any term is by default set to 100. This implies that a value of 25 indicates the search term is one fourth as popular and a value 0 indicates that there is no data available for a particular term. Figure 1.3 shows that the search interest for Apache Spark has significantly outpaced Apache Hadoop as most popular big data processing framework since mid of 2014.

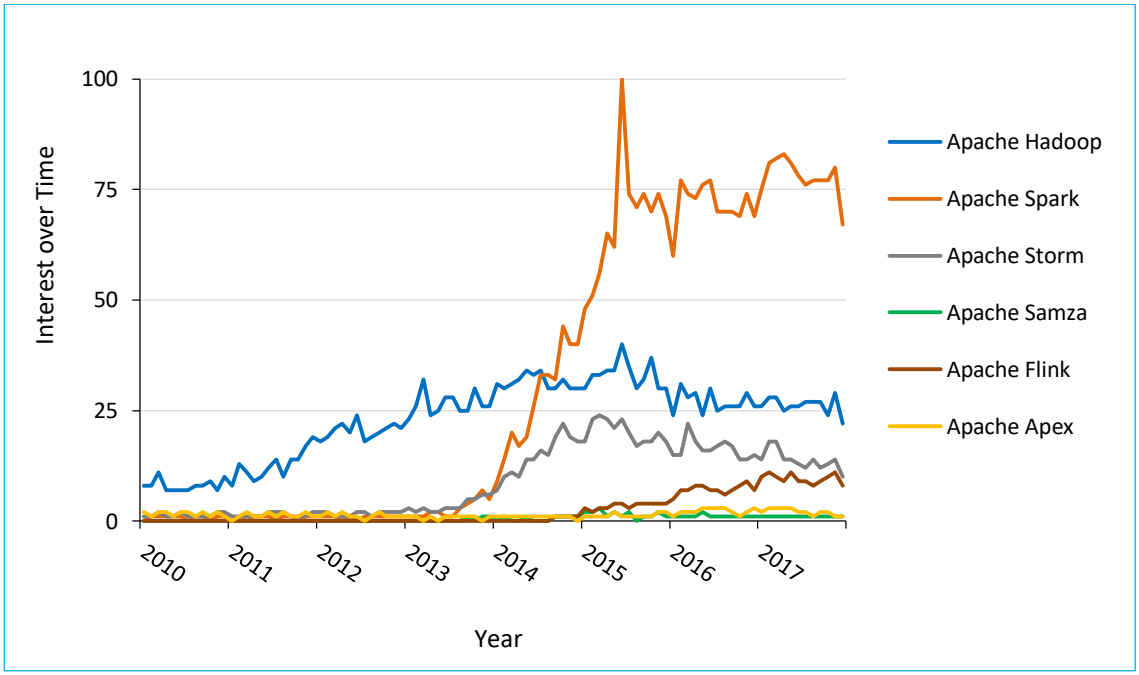


Figure 1.3: Google worldwide search interest starting from 2010 to 2017 for big data processing frameworks.

1.5 Major Research Challenges

Though there is a lot of hype about big data, yet it is underutilized due to numerous challenges which big data technologies present. Big data research is necessary to find out the practical and innovative solutions to address such challenges. It adopts an interdisciplinary approach aiming to address a wide range of real-life problems using data-driven models. This section discusses the major research challenges pertaining to big data.

1.5.1 Data Acquisition and Metadata

Data sources are generating a huge volume of data and most of it is noisy data. Data acquisition is a process of collecting, filtering and cleaning the data before storing it in a data warehouse or some other storage device. The data has to be filtered and compressed so that it does not throw away any relevant information. Defining such filters is one of the major challenges. Another challenge is to enable automatic generation of the metadata. Metadata describes what data is logged, and how it is logged and measured. New methods and approaches are needed to record the context and semantics of the data.

1.5.2 Data Quality

Big data is generated from diverse heterogeneous sources, which is mostly of different types. One of the key characteristics of big data - veracity, deals with completeness, accuracy and quality of data. Better insights depend upon data quality. More than often, data contains mistakes, errors, and incomplete values. Poor data quality has become a serious problem. According to [43], it costs US economy around 3.1 trillion dollars per year. Therefore, it is pertinent to overcome the bad data quality quickly. Understanding of data semantics and corrections is a challenging task. In many data warehousing projects, the data cleaning is a tedious task and it consumes around 30-80% of the development time and budget. With these numbers, one can imagine the impending challenges and opportunities to improve data quality [44]. The absence of data quality management can result in financial losses, inefficiency, and failure to comply with rules and regulations [45].

1.5.3 Data Storage

The digital world will keep on outpacing the storage capacity. In 2013, it was just capable of holding 33% of the digital world and by 2020, the percentage will fall to less than 15%. Fortunately, most of the digital world is transient (like streaming and media data) and it is becoming more so [3]. For high-speed data, it is necessary to have low latency and very fast input-output rate. Traditional hard disk drives (HDDs), a primary storage option, are mainly used to store the persistent data, but cannot meet the challenge due to slow provision and lack the performance. The HDDs will be replaced gradually with solid state drives (SSDs). A new approach, software-defined storage (SDS) provides better flexibility, scalability, agility and use of resources. It separates programming (to control

storage tasks) from hardware (to store data). Moving to a new storage system is another challenge as it affects every aspect of data processing.

1.5.4 Data Sharing and Transfer

Data should be made available and accessible for research and development to realize its maximum value and impact. The data sharing includes source data, metadata, any standards and the APIs to the data. It helps to draw better conclusions which are based on data reuse and combination of the original dataset with other datasets for different purposes. Though data is growing tremendously, yet most of it remains inaccessible or dark. Sharing of big data becomes a challenging task due to its heterogeneity and scale. To make effective use of available data, it is necessary to develop new technologies, practices and standards for data sharing across various disciplines. Making open-source data accessible for all, like open-source software, would be useful. Further, moving petabytes or terabytes of data seamlessly from one location to another is a potential challenge. One time transfer of massive data seems manageable, but doing regular updates or sharing is a challenging task. The solutions like data streaming, micro-batching and data pipelining may be considered for data transfer.

1.5.5 Scalability

Scalable solutions are required to handle size, speed and complexity of big data. Computing systems should be scaled up to tackle such aspects of big data and the corresponding solutions should be highly interactive, easy to use, resilient and highly fault tolerant. In order to interactively process high-speed data at a large scale, very fast interconnected networks, large system configurations, deep memory hierarchies, and high-performance storage systems are essential. Simple scaling of existing tools and technologies is not sufficient [46].

IoT, smart devices connected to the Internet, is also unleashing a new wave of research challenges and opportunities. It is expected to contribute over half of Internet traffic by 2018 [47]. The data from heterogeneous sensors and networks will provide a huge amount of highly complex data. State-of-the-art big data technologies will be essential for handling such complex data. More statistical approaches should be incorporated in computer science techniques, though it is challenging to have scalable statistical models when data becomes too large [48].

The relational databases are not supposed to deal with the scale and agility challenges pertaining to big data applications. NoSQL (Not only SQL) databases not only meet such demands but also take advantages of the commodity hardware and immense processing power. For obvious reasons, NoSQL databases are more scalable, easy to use, flexible and provide better performance as compared to relational databases. As of now, MongoDB is the most popular NoSQL databases followed by Apache Cassandra, Redis, and HBase. Neo4j and Amazon DynamoDB are the most popular NoSQL graph database and cloud database respectively [49, 50]. Table 1.8 provides various categories of NoSQL databases.

Table 1.8: Various categories of NoSQL databases.

NoSQL Database Types	Performance and Scalability	Flexibility	Complexity	Examples
Wide-Column Store	High	Moderate	Low	Apache Cassandra, Apache HBase, Google Bigtable, Vertica.
Document Database	High	High	Low	MongoDB, RethinkDB, Azure DocumentDB, Terrastore, Apache CouchDB.
Key-Value Store	High	High	Very Low	Amazon SimpleDb, Azure Table Storage, Oracle NoSQL Database, Linkedin Voldemort, Berkeley DB, RocksDB, Redis, Riak.
Graph Store	Very High	High	High	Neo4j, AllegroGraph, GraphBase, Cayley, Dgraph, GraphDB, JanusGraph, InfiniteGraph.

1.5.6 Data Analysis

There are three main attributes of big data - the data, the analytics, and the visualization of the results, which is easy to interpret. The term analytics is used for any data-driven decision making or statistical and mathematical data analysis to predict most happening scenario. Data analytics is a kind of workflow which reduces the massive amount of low-value data down to a small size of high-value data in order to extract actionable sense out of it. In many applications, the data analysis is a bottleneck because of the data complexity as well as the lack of scalability of the underlying algorithms. With the availability of big

data, the decision making can be improved in the critical domains like healthcare, natural disaster, crime and security, employment, economy and education [51]. As big data technologies and techniques are gradually evolving with time, paving the way for better insights, there is an apparent shift from 'Information Societies' to 'Knowledge Societies' [52].

In the last few years, data analytics has shown impressive growth due to the decrease in the cost of data acquisition, enabling collection of fine-grained data. The low-cost storage has also helped to retain a huge volume of data in the hope of getting better actionable insights [53]. A few key characteristics of big data analytics are: (a) exploring semi-structured and unstructured data to find out actionable insights; (b) achieve near real-time analytics by minimizing time gap between data acquisition and analysis; (c) enable deep analytics; (d) looking for low cost and truly scalable data analytics frameworks [54].

At present, only a small fraction of the digital universe can be analysed using big data technologies. In 2013, only 22% of digital information (i.e. 643 EB) was useful if it were tagged to have metadata. In fact, just 5% of the potentially relevant data was tagged, and even less was analysed. By 2020, around 37% of digital data (more than 13,000 EB) would be useful for analysis mainly due availability of more data from embedded devices, yet big data technologies shall only be able to analyse around 10% of the useful data enabling extraction of actionable insights or intelligence out of it [55]. It means that a huge chunk of information will remain unprocessed in coming years due to unavailability of relevant big data technologies. It is one of the major research challenges to convert this huge digital information into insightful knowledge using sophisticated big data tools and techniques.

1.5.7 Data Querying and Indexing

In big data querying and indexing, there are several challenges. First, big data is stored in a distributed system and therefore, the index structure and query theory for large datasets should be developed based on such systems. Second, the tree-like structure, which is highly prevalent in traditional querying optimization and indexing techniques for small datasets, does not work well with large datasets. Third, fault tolerance is an important aspect which needs attention in big data query and indexing [56]. Besides, maintaining a reasonable query performance and throughput in big data analytics is a significant research challenge.

1.5.8 Data Uncertainty

Data uncertainty is crucial when data is collected from multiple heterogeneous sources. Usually, there are abrupt changes in data uncertainty with the increase in the size of data, which cannot be handled by traditional data processing tools. It affects the performance of learning from big data. In recent years, the researchers are trying to integrate the data uncertainty with learning from big data [57]. To ensure the reliability of the insights drawn from big data, suitable approaches are required to understand data uncertainty.

1.5.9 Data Privacy, Security and Ethics

Data privacy, security and ethics are three major concerns in the big data innovation ecosystem. Data privacy concerns about how information is seen and managed; security concerns about the protection of personal or critical information; and ethical concerns about misuse of the data analysis. Privacy is one of the biggest worries and the right to privacy has led to many new research trends [58]. In today's interconnected and open world, more and more information is shared including personal details and the dark side of big data is that it allows scrutiny of such information. There are no clear guidelines and standards for protecting individual privacy. The inferences like social media knows you are pregnant before your father knows, are threatening to breach thin boundaries of privacy. In short, data privacy is a technical as well as a sociological problem [59]. There is a need to distinguish privacy research from security research. Weak cybersecurity is a threat to privacy, though even with robust cybersecurity, there may be privacy violations [60].

There is a need to develop tools and technologies for securing large-scale data in a highly distributed network [61]. Big data ethics are required to avoid undue benefits to specific sections. While data sharing is essential, defining data privacy and security are very critical in all sharing platforms. Trust in privacy as well as security measures is an integral part of any data-sharing technology. Moreover, it also handles the concerns about ethics as well as social implications of big data.

1.5.10 Data Visualization

Data visualization plays a vital role in data analysis process and helps in revealing the useful insights. Moreover, the visualization based data discovery facilitates easy to use, dynamic and interactive analytical views. However, the visualization of big data at rest

or in motion is one of the major research challenges. The main challenges are related to perceptual, real-time, and interactive scalability. These challenges can be addressed by reducing data and latency [62]. With automatic techniques still evolving, interpretation of analysis may require human involvement resulting in next-generation visualization techniques. Such new approaches are expected to evolve over the time.

1.6 Thesis Outline

The organization of the thesis is described below:

Chapter 1 discusses basic concept of big data, its evolution including a bibliometric study of academic and industry publications during the period 2000–2017, and popular open-source big data stream processing frameworks. The chapter also discusses prevalent research challenges which must be addressed to realise the true potential of big data. This chapter partly addresses objective 1.

Chapter 2 describes the relevance of big data analytics, machine learning and deep learning in cybersecurity. In addition, it discusses popular deep learning open-source tools. The chapter elaborates the use cases, examples from the literature, and existing popular solutions of big data security analytics. A significant portion of the chapter discusses the background information of malware detection techniques which has been considered as a use case for this research. An introduction to different approaches to malware analysis along with their pros and cons are discussed. Finally, it provides the review of related research work. This chapter accomplishes objective 1.

Chapter 3 lists the research gaps that have been identified while carrying out literature review. The chapter presents the problem statement, lists the research objectives and concludes by defining the brief methodology for achieving these objectives. This chapter partly accomplishes objective 2.

Chapter 4 presents the design of the proposed architecture for understanding malware behavior trends from year 2010 to 2017. It discusses the steps to prepare malware dataset by setting up an automated malware analysis environment using distributed Cuckoo Sandbox with Windows 7 as a guest machine. Further, various open-source tools used to implement the framework have been described. The chapter also discusses the methodology and evaluation results obtained though statistical analysis of malicious

binaries which can be shared with the stakeholders to focus on the prevalent areas. This chapter partly accomplishes objective 2.

Chapter 5 presents a proposed framework built on the top of Apache Spark for zero-day malware classification using machine learning. An integrated set of static and dynamic features is created by extracting the relevant information from the JSON reports. Different machine learning models are built for classification of malware and compared to choose the best model for our dataset on the basis of evaluation metrics like false positive rate, false negative rate and accuracy. This chapter partly accomplishes objectives 2 and 3.

Chapter 6 presents the proposed methods based on ensemble learning using big data architecture for improving the performance of malware detection. The first method is based on the weighted voting strategy of ensemble learning and the second method selects an optimal set of base classifiers for stacking purpose. The proposed solution is tested and evaluated on a dataset consisting of integrated set of features obtained after performing static and dynamic malware analysis. The experimental results show that the proposed methods improve the generalization performance in detecting new malware as compared to traditional ensemble methods. This chapter partly accomplishes objectives 2 and 3.

Chapter 7 presents the proposed solution for malware detection by bringing together the benefits of Apache Spark – a big data processing engine, and Keras along with TensorFlow, a deep learning framework. Apache Spark is deployed for efficient preprocessing of data, and Keras along with TensorFlow is used to develop and train deep neural network. The resulting model is trained and evaluated on a dataset having integrated set of malware static and dynamic features. The resulting classification model gives better performance than the traditional machine learning algorithms. This chapter accomplishes objectives 2 and 3.

Chapter 8 concludes the thesis by providing a brief overview of the proposed framework and provides insights about the future scope of work. It highlights the contributions made and enlists the points which can be undertaken in future.

1.7 Summary

This chapter presented the basic taxonomy of big data and its evolution. It presented a bibliometric study of academic and industry publications during the period 2000–2017,

and popular open-source big data stream processing frameworks along with their comparison. It also presented the prevalent research challenges which must be addressed to realise the true potential of big data.

Chapter 2: Background and Related Research Work

In today's era of digitization, a huge amount of data being generated has resulted in an exponential growth of widespread cyber threats in real-time environments like cyber-terrorism, computer & network intrusion, espionage, and cyber frauds. From the large-scale retailers to healthcare providers to government agencies, nothing remains safe from today's sophisticated and targeted cyber-attacks. Moreover, the ever-evolving threat landscape and rapidly growing network environments are offering additional ways for the attackers to break in. This scenario has overwhelmed the existing traditional solutions and rendered them outdated to handle such attacks. To encounter the real-world cybersecurity challenges, the security researchers are putting a lot of effort on technologies stemming from areas like big data, machine learning, and artificial intelligence to extract powerful insights triggering the incident response against malicious and suspicious data.

This chapter describes the relevance of big data analytics, machine learning and deep learning in cybersecurity. In addition, it discusses popular deep learning open-source tools. The chapter elaborates the use cases, examples from the literature, and existing popular solutions of big data security analytics. A significant portion of the chapter discusses the background information of malware detection techniques which has been considered as a use case for this research. An introduction to different approaches to malware analysis along with their pros and cons are discussed. Finally, it provides the review of related research work. This chapter accomplishes objective 1.

2.1 Introduction

In recent years, there is a disturbing trend - attackers are innovating faster than defenders. Due to the evolution of modern techniques like cloud computing, IoT, social networking

and mobile devices, the cyber attackers are provided with the opportunities to explore and exploit the vulnerabilities for creating sophisticated attacks. The sophistication and innovation lead to a change in focus of cyber-attacks. Earlier, the focus was mainly on economic espionage and now it is becoming politically motivated affecting and shaping the governments causing unprecedented level of disruption [63]. The high availability of sophisticated tools has made the cyber-attacks more directed, persistent and unknown. These can easily evade traditional defences and stay undetected on the network of the target organization. The traditional cybersecurity solutions make use of signature-based methods and thus, are not able to detect the advanced cyber-attacks/threats which are making use of new techniques and technologies.

Intel [64] estimated that there will be 200 billion IoT smart devices (i.e., 26 smart objects for every human being on earth) by 2020. The data from IoT devices (like sensors, system monitoring physical universe) will contribute over half of Internet traffic by 2018 [47] and is expected to grow to 10% by 2020 [3] making it attractive to attackers. Vulnerabilities in cloud services provide next frontier of cybercrime. Many IoT devices rely on cloud services/apps to store personal data, which can be accessible to the attackers allowing them to exploit the cracks in clouds. The increasing use of IoT and cloud services is making handsome contributions to the growth of cyber-attacks. Moreover, the ever-growing digital world which is nearly doubling its size after every two years [3] and is likely to grow to 163 ZB by 2025 [65], will further lead to more cyber threats. All these staggering statistical figures depict the tremendous growth of security data making cybersecurity a big data problem which poses enormous challenges.

As a response to abovementioned problems, a new generation of security analytics solutions known as big data security analytics (BDSA) have started evolving. These solutions can ingest, store, process and analyze vast threat data helping organizations to detect the attacks in real-time. The data is analyzed by using artificial intelligence, machine learning, and deep learning algorithms enabling quick detection of anomalies. Such emerging solutions are different from traditional security information and event management (SIEM) systems as these function in near real-time and produce a few security alerts rated by their severity level associated with underlying risk model. These solutions make use of big data technologies, which have become cornerstones of digital transformation, and are now more accessible and cost-effective mainly due to the availability of commoditized hardware and open source tools. More and more

organizations are adopting such new approaches to catch up with the latest trends in cyber-attacks allowing them to shed old reactive approach and adopt a proactive approach, i.e., to shift from traditional cybersecurity to cyber threat intelligence (CTI). The basic idea behind CTI is to have the ability to recognize the indicators of attack gradually as it progresses, putting together the shared information about attack methods, and act in a timely fashion. These emerging solutions have started to show the signs of future potential.

2.2 Big Data Analytics

Big Data has changed the landscape of security analytics. In many applications, the data analysis is a bottleneck because of data complexity as well as non-scalability of underlying algorithms. Thus, there is a need to develop more big data tools and techniques. There are three main attributes of big data - the data, the analytics, and the visualization of the results. The term analytics is used for any data-driven decision making or statistical and mathematical data analysis to predict most happening scenario. Figure 2.1 depicts four types of data analytics which are discussed below:

- Descriptive analytics analyses the historical as well as real-time data to come across insights like reasons for success or failure of any past event. This category covers a majority of big data analytics.

- Diagnostic or behavior analytics is probabilistic in nature and emphases on uncovering the issues, factors and incidents responsible for any outcome.

- Predictive analytics is a form of advanced analytics which is built upon trends and patterns located in historical as well as transactional data to know what might happen in future.

- Prescriptive analytics is the most useful advanced analytics to make something happen by achieving the best outcomes and better decisions. It decides about the possible actions which can be taken by using optimization techniques like stochastic optimization. It is comparatively complicated as it makes use of complex mathematical modelling including AI, deep learning, machine learning, natural language processing, statistics, operations research, etc. At present, very few organizations possess the capability to use prescriptive analytics.

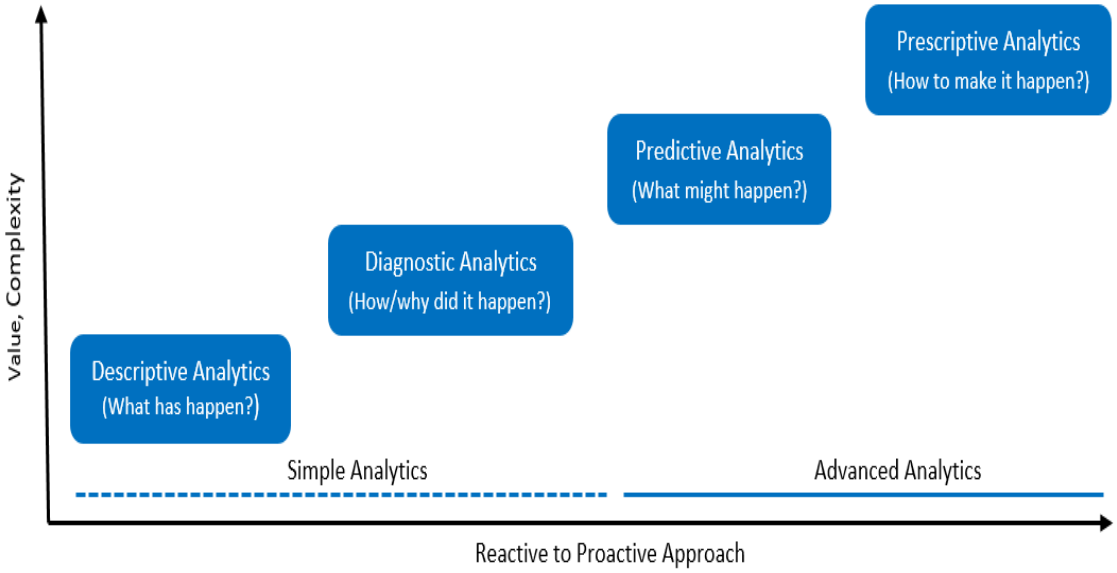


Figure 2.1: Types of data analytics.

In general, BDA is a multi-step process [54] as shown in Figure 2.2. The first step is data acquisition from diverse sources. The data sources can produce massive amount of raw data comprising of around 80% unstructured data. Moreover, most of the data is noisy and thus, it needs to be filtered and compressed. It is a challenge to create such filters that do not lose any useful information. Further, the format of the information accumulated cannot be analyzed directly. It is extracted and expressed in a structure suitable for analysis. The final step is visualization, which is one of the simplest ways to quickly understand the meaning of the data. For efficient large-scale analysis, these steps should be automated which demands automatic resolution of various data structures and semantics.

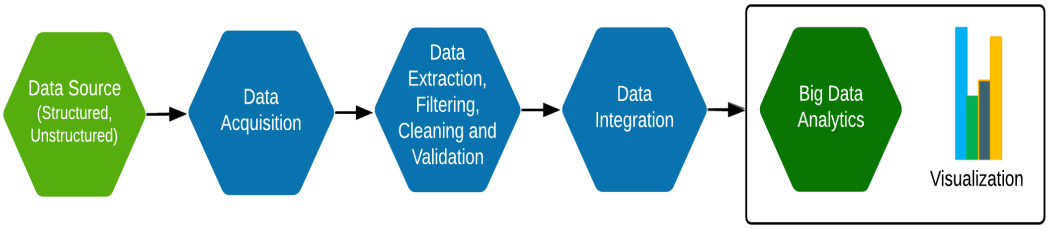


Figure 2.2: Big data analytics pipeline.

In last few years, BDA has caught much attention of security researchers, practitioners and policymakers. It deploys machine learning algorithms and other decision-making techniques, along with augmenting human interface, to generate useful patterns which can further be used to thwart future cyber-attacks. Apart from streaming data, BDSA also uses historical data, which streamlines primary calibration to the regular patterns of any activity in order to detect anomalies and thus, can augment CTI to deal with future threats. Using big data frameworks like Apache Hadoop, Apache Spark, etc. and inexpensive/commodity hardware, many organizations have attained the capabilities to store and analyze large unstructured threat data in real-time to accomplish the promise of CTI in today's threat landscape.

2.3 Big Data Analytics, Machine Learning, and Deep Learning in Cybersecurity

Due to the ever growing advanced, persistent and sophisticated cyber-attacks, the rule and signature-based traditional prevention solutions have failed miserably to detect or stop such security attacks. Moreover, these require manual intervention and are limited to descriptive and diagnostic analytics, which fail to respond to the challenges of big data. To deal with this problem, new security analytics solutions have been developed recently, which can collect, store, process, and analyze a large amount of unstructured threat data in real-time. Additionally, by using the context data and external threat intelligence, these solutions can integrate big data technology and advanced analytics tools to correlate security events from multiple data sources for providing early detection of malicious activities in real-time. The alerts generated are ranked by severity and include forensic details which help the security analysts in early detection and mitigation of advanced cyber threats.

Machine learning is next big thing in big data. In recent times, the innovations in machine learning are staggering. With the proliferation of big data, machine learning has become the forefront of research and industry security applications. The big data insights converging from predictive analytics involve machine learning. In general, machine learning is used to devise complex models, data structures, algorithms that lead to useful actionable predictions. At the same time, machine learning has never been challenged before as the traditional machine learning algorithms face many critical challenges to transform big data into actionable intelligence. BDA along with machine learning form

the first line of defence in cybersecurity to predict, detect, prevent and respond security violations. Machine learning identifies anomalies, correlations, trends, and patterns leading to intelligence discovery. It also helps to remove noise from enormous amount of data and can lead to more sophisticated insights using automated predictive analytics. More than half of the web traffic has already been encrypted resulting in better security. At the same time, it provides more opportunities for invaders to conceal spurious activities causing more damage. Thus, more and more security solutions are turning to AI and machine learning to identify the unusual patterns in encrypted environment [66]. The applications of BDA in the field of cybersecurity is just in the fledgling state. According to Gartner [67], BDA will play a crucial role in cybersecurity and by 2018, 25% of security products used for detection will have some form of machine learning built into them.

Recently, there is an exceptional growth in the advances in deep learning [68] in various domains including information security. It mimics the human brain and processes the information at multiple levels of abstractions from raw data [69]. These models have the capability to learn complex and non-liner functions though multiple layers for making decisions rather than relying on domain experts for crafting features [70]. Deep learning has revolutionized fields like computer vision, speech recognition, human language translation, healthcare, finance, retail, transportation and logistics, manufacturing, media and entertainment, travel and tourism, agriculture, energy [71] and resources, cybersecurity, and education, and many other fields. It has already a fairly lengthy list of inter-disciplinary applications like forecasting floods, predicting risk of cardiac events, tracking fishing activity in real-time, helping farmers to identify diseased plans, monitoring and protecting endangered wildlife species, detecting deforestation and so on.

Until recently, deep learning has been relegated to the theoretical realm only. The enormous growth in computational resources, the convergence of algorithmic advances, availability of easy-to-use open-source libraries, and perhaps most important of all, the data deluge, have made it possible to realize practical aspects of deep learning. The availability of big data exposes deep learning algorithms to more complex and high-dimensional data allowing it to identify even narrowly defined patterns. Most of the big data comprises of semi-structured or unstructured data. Deep learning performs exceptionally well with such data as compared to traditional machine learning algorithms. In addition, with the increase in volume of training data, the performance of deep neural

networks (DNNs) tends to increase whereas the traditional analytics techniques hit the plateau after certain point. For example, a supervised DNN attains satisfactory performance with around five thousand samples per category, but can attain or surpass human performance with a training dataset of ten million or more samples [72]. However, if data volume is thin or below threshold, there won't be any value addition to the traditional data analytics. It is pertinent to understand that the performance can be achieved up to a point as either the model runs out of training data or neural network becomes so large that it takes too long to train.

2.3.1 Scalable Machine Learning Libraries

The popular big data frameworks like Apache Hadoop, Apache Spark, Apache Storm, Apache Samza, Apache Flink, and Apache Apex make use of either inbuilt or common machine learning libraries.

Apache Mahout is a library for scalable machine learning applications running on the top of Hadoop and automatically finds patterns available in datasets stored in Hadoop Distributed File System (HDFS). It provides the implementation of many learning algorithms like classification, clustering, regression and dimensionality reduction, etc. It supports Apache Spark, H2O and Apache Flink. It also includes a math environment known as Samsara, which includes general linear algebra, statistical operations and data structures.

Apache Spark MLlib [73] is a scalable machine learning library built upon Apache Spark. The iterative computations of Apache Spark makes it suitable of large scale machine learning applications. MLlib consists of fast and scalable implementations of standard machine learning algorithms like classification, regression, clustering, collaborative filtering, and dimensionality reduction. MLlib provides a high-level API in Scala, Java, R and Python to leverage Apache Spark ecosystem. It can be used with any data source like HDFS, HBase, or local file.

Apache SAMOA (Scalable Advanced Massive Online Analysis) is a distributed streaming machine learning framework and is presently undergoing incubation at Apache Software Foundation (ASF) [74]. It provides a collection of common machine learning algorithms and supports the development of algorithms without worrying about the complexity of underlying distributed stream processing engines (DSPEs) like Apache

Storm, Apache Samza, Apache Flink, and Apache Apex. Once implemented, such algorithms can run on any DSPEs due to its pluggable architecture. In general, it is similar to Mahout, but specifically designed for stream mining.

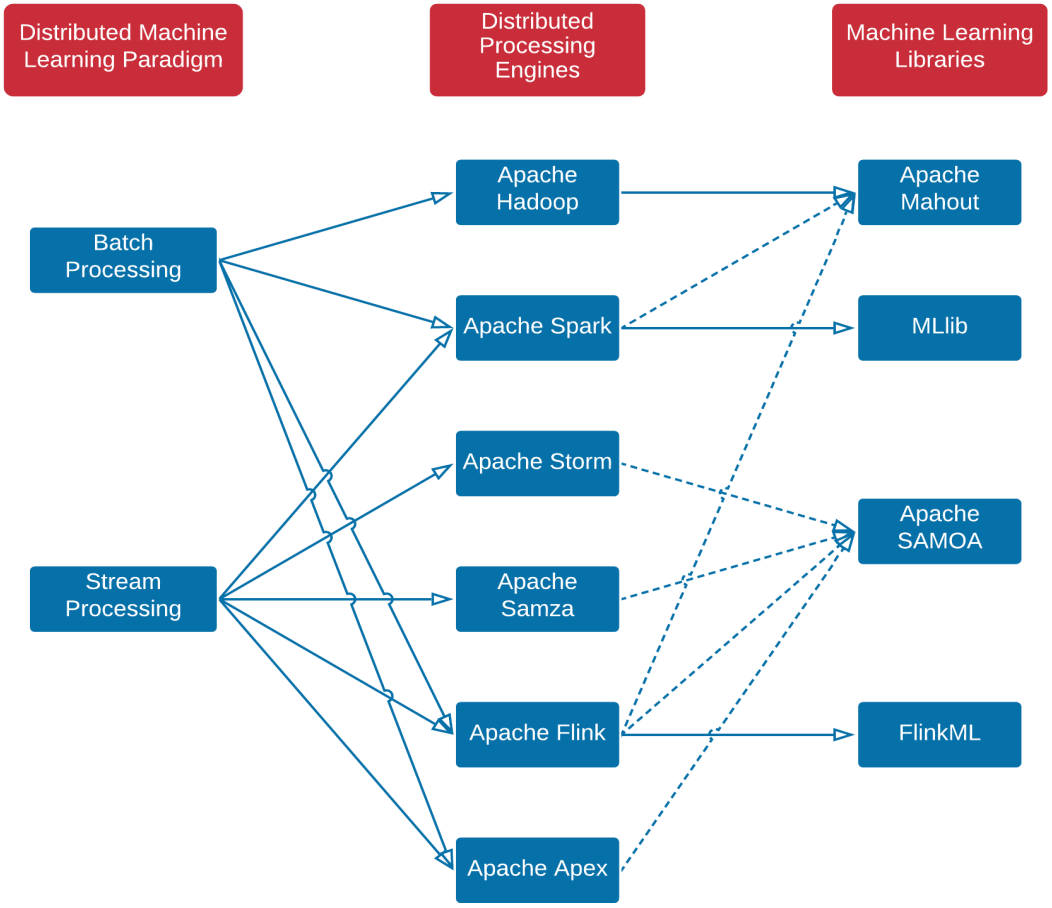


Figure 2.3: Distributed machine learning paradigm along with big data processing engines and their machine learning libraries.

FlinkML is the machine learning library for Flink. It is an easy to use library. Presently, it supports algorithms like supervised learning, unsupervised learning, data preprocessing, recommendation, outlier selections, and utilities like distance metrics and cross-validation. Machine learning systems developers tend to write a lot of glue code during its implementation [75]. FlinkML helps developers keep glue code to a minimum. Figure 2.3 depicts distributed machine learning paradigm along with big data processing engines and their machine learning libraries. The solid lines describe direct dependency

and the dotted lines represent the compatibility of a particular machine learning library with a distributed processing engine.

2.3.2 Open Source Deep Learning Frameworks

The deep learning algorithms are commonly implemented at scale using higher-level frameworks so that deep learning model development becomes simple and efficient. These frameworks raise the programming abstraction making it easier to map the program to the problem architecture. The keen interest of top organizations in deep learning led to development of many frameworks. In this section, we discuss the most widely used open-source deep learning frameworks along with their comparison in Table 2.1.

2.3.2.1 Tensorflow

TensorFlow [76] is an immensely popular end-to-end open source platform for large scale distributed training and inference. It offers easy model building, robust production anywhere and a powerful experimentation tool for state-of-the-art research models. It is used for graphical computational model in which the nodes and edges denote mathematical operations and multidimensional data in the form of tensors (i.e. n-dimensional array or list) respectively. TensorFlow computations become too complex for tasks like training massive DNN and thus, it includes a suite of visualization tools, TensorBoard, for easy understanding, debugging, and optimization of programs. Another tool, TensorFlow Serving, helps in easy deployment of models in production environments and yet it retains the same server architecture and APIs. TensorFlow's flexible architecture helps in easy deployment to multiple CPUs, GPUs, TPUs in a desktop, server, cluster or mobile environment without rewriting code. It has a rich documentation and community. The use cases include variety of applications like speech recognition, Google search, Google Translate, Gmail, and many more.

2.3.2.2 Keras

Keras [77] is a high-level API to develop and train deep learning models using TensorFlow, Microsoft Cognitive Toolkit (CNTK), or Theano as its backend engines. It is very user friendly, modular, and extensible. The models are described in Python which is compact and easy to debug. Being an open-source library, Keras API is very popular among researchers and developers for enabling fast experimentation and efficient production. It supports CNN, RNN, as well as combination of both.

Table 2.1: Comparison of popular open-source deep learning frameworks.

Comparison Metrics	**TensorFlow**	**Keras**	**PyTorch**	**Apache MXNet**	**CNTK**	**Chainer**
Released on	November, 2015	March, 2015	October, 2016	2015	2012	June, 2015
Software License	Apache 2.0	MIT	BSD	Apache 2.0	April, 2015 (MIT)	MIT
Most Stable Release (as of April, 2019)	1.13	2.2.4	1.0	1.4.0 (incubating)	2.6.0	3.2.0
Creator	Google Brain	François Chollet	Facebook	ASF	Microsoft Research	Seiya Tokui
Platform Supported	Linux, Mac, Windows, Android	Linux, Mac, Windows, Android	Linux, Mac, Windows	Linux, Mac, Windows	Windows, Linux	Windows, Linux
Implemented in	C++, Python	Python	Python, C, C++	C++, CUDA	C++	Python
Client Language Interface	Python, R, C++, C#, Java, Go, Julia	Python, R	Python, C++	Python, C++, Java, Julia, Clojure, Perl, R, Scala	Python, C++, C#, Java, BrainScript	Python
Static / Dynamic Computational Graph	Static	Static	Dynamic	Static	Static	Dynamic
Programming Style	Symbolic	Symbolic, Imperative	Imperative	Symbolic, Imperative	Symbolic	Imperative
Debugging	Difficult	Average	Easy	Easy	Easy	Easy
Distributed Training	✓ (Horovod)	✓ (Horovod)	✓ (Horovod)	✓ (Horovod)	✓	✓
Multiple GPUs Support	✓	✓	✓	✓	✓	✓
Scalability	Average	Near-linear	Near-linear	Near-linear	Near-linear	Near-linear

Visualization	✓ (TensorBoard)	✓ (TensorBoard)	✓ (Visdom, tensorboardX)	✓ (Jupyter Notebook, Graphviz)	✓ (TensorBoard, Graphviz)	✓ (ChainerUI)
Pre-trained Models	✓	✓	✓	✓	✓	✓
RNN, CNN Support	✓	✓	✓	✓	✓	✓
Customizable	×	×	✓	✓	✓	✓
Cloud Support	✓	✓ (Google Cloud Platform)	✓ (AWS, Google Cloud Platform, Microsoft Azure)	✓ (AWS, Microsoft Azure)	✓ (Amazon EC2 and Azure)	✓ (Amazon EC2 and Azure)
Open Neural Network Exchange (ONNX)	×	×	✓	✓	✓	✓

2.3.2.3 PyTorch

PyTorch (https://pytorch.org/) is an end-to-end deep learning platform for rapid prototyping and fast experimentation, scalable distributed training, and efficient production. It supports automatic differentiation [78] and has a rich ecosystem of tools and libraries for building deep learning models, which can be exported to other open neural network exchange (ONNX) compatible platforms. In addition, the support for cutting-edge AI models like dynamic computation graphs has made it popular among AI researchers. The availability of PyTorch on popular cloud platforms like AWS, Google Cloud Platform, and Microsoft Azure, helps in providing frictionless development, large scale training and production. Its hybrid front-end offers easy switching between eager and graph mode providing more flexibility and performance. In addition, the C++ front-end is suitable for high performance and low latency applications. Recently, Facebook has merged Caffe2 – an open-source, lightweight, modular, and scalable deep learning framework, with PyTorch to create a unified framework.

2.3.2.4 Apache MXNet

Apache MXNet [79] is a lightweight, multi-language, portable, programmable and scalable deep learning framework. It speeds up the development and deployment of large scale DNN by exploiting the full capabilities of GPUs and cloud computing. MXNet offers near-linear scalability for distributed training depending upon number of GPUs in a cluster. It supports a set of programming languages giving choice and flexibility for the developers, though all code is compiled in C++ irrespective of the language used to build a model and thus yields better performance. Gluon, the imperative interface of MXNet, is a clear, concise, and simple API for deep learning which helps in easy prototyping, building, and training of deep learning models without affecting the training speed. The key advantages of Gluon includes simple and easy-to-understand code, flexible and imperative structure, dynamic graphs and high performance. In just a few lines of code, Gluon can build linear regression, CNN, recurrent LSTMs for applications like image recognition, speech recognition, forecasting, and natural language processing (NLP). In version 1.4.0, MXNet incorporated Horovod [80], open-source distributed framework created at Uber, for fast and easy-to-use distributed training of models.

2.3.2.5 Microsoft Cognitive Toolkit (CNTK)

CNTK [81] is an open-source, production-ready, and efficient toolkit for distributed deep learning which can scale across multi-GPUs and servers. It provides fast experimentation of feed-forward DNN, CNNs, and RNNS/LSTMs in text, speech and image recognition. CNTK is mostly implemented in C++ which enables it to deliver better performance [82]. Around 80% of Microsoft's internal deep learning workload runs on CNTK. It supports ONNX format for framework interoperability and shared optimization. On the downside, it has a relatively small community.

2.3.2.6 Chainer

Chainer [83] is a Python based versatile and intuitive deep learning framework which uses define-by-run scheme (i.e. dynamic computational graphs) and object oriented high-level APIs to build and train models. It is faster than other leading Python frameworks. It requires just a few lines of code to leverage a GPU and provides parallelization for multiple GPUs. Chainer has five additional domain specific libraries - ChainerMN (multi-node), ChainerRL, ChainerCV, ChainerUI, and Chainer Chemistry, for distributed deep learning, deep reinforcement learning, computer vision, visualization and experiment management, and graph convolutions for biology and chemistry tasks respectively. Currently, Chainer is supported by leading organizations like IBM, Intel, Microsoft, Nvidia, and AWS.

2.4 Big Data Security Analytics

In cybersecurity, the primary focus is on detection of any attack, which is possible by recognizing the changing patterns, analyzing the massive data close to the real-time, and performing correlations among the data emerging from diverse sources like server/application logs, network events, user activities, etc. More recently, BDA has been used in cybersecurity where the data is complex (collected from different sensors and social media) and large (ranging from terabytes to petabytes) and thus requires the advanced storage & processing methods, analysis techniques and visualization technologies.

Combining big data with security analytics lead to security intelligence, which aims early identification of targets and security threats. BDSA deploys techniques from BDA and machine learning to generate insights from a large amount of data to prevent cyber-attacks

and frustrate the cyber attackers [84]. It can enhance cyber-attack resistance by better threat identification and response in real-time, which is one of the most effective defences against cyber intrusions. In addition, it helps to reduce the time taken for threat detection, and uncover hidden relationships, patterns, and ultimately, remove security threats. The information needed to understand the threat pattern loses its value over time and thus, cybersecurity analytics must combine real-time analytics on data in motion with analysis of historical data at rest. This combination can provide invaluable insights leading to threat intelligence, which can predict and mitigate real-time cyber-attacks automatically. BDSA can be used to analyze log files, identify suspicious activities in network traffic, correlate information from heterogeneous sources, etc. to improve the information security and situational awareness.

In general, BDSA is different from traditional SIEM systems (see Table 2.2), which are used to provide quick identification, analysis and recovery of the security event. SIEM combines security information management (SIM) and security event management (SEM); the former helps to identify the threats as a real-time process (as much as possible) while the latter deals with capturing log and related information for subsequent analysis. Unlike traditional SIEM solutions, BDSA leverages sophisticated machine learning algorithms to accurately identify the most hard-to-detect cyber threats, insider threats and fraud.

2.4.1 Key Features

The key features that make BDSA different from other domains of information security are:

- **Flexible Architecture**

BDSA possess the ability to collect, store, process, and analyze massive data for a range of security analytics activities. The architecture tends to be flexible so that it can be scaled linearly to fulfill the security prioritization of the organizations.

- **Analyze Diverse And Complex Data**

The security data sources where the BDA can be applied include not only the active (real-time) sources like social network data, virtualized cloud networks, biometric data, digital certificates, credential information, but also the passive sources like mobile data, data related to SIEM, computer-data, clickstream patterns, etc. The integrated data collected

from these active and passive sources can be easily correlated and analyzed to find insights about the attackers' behavior which help to take preventive measures to deal with similar future threats [85].

- **Persistent Big Data Storage And Performance**

BDSA platforms make use of big data storage systems and have the capability to include the computational requirements of security analytics (for processing data analytic algorithms and complex queries in near real-time).

- **Threat Intelligence Feeds**

BDSA platforms must have the capability to ingest data from a variety of sources and correlate these to generate insights for updating the vulnerability databases, security blogs and other related sources having valuable information. It identifies the advanced threats as well as anomalies which need immediate investigation.

- **Contextual Information**

Contextual information can include data in the context of users, devices, events, and dynamic patterns. Such information gives better behavior analysis and anomaly detection. The absence of context-rich data may lead to high false positives.

- **Real-Time And Historical Data Analysis**

Real-time and historical data analysis over a period can identify any deviations from normal behavior/trends. It helps in reducing statistical noise among data and false threat alerts. By filtering out the statistical noise, BDSA reduces the vast amount of security alerts into a few security alerts rated on the basis of their severity level. The real-time actionable information provides better risk management and incident detection/response. Retaining historical information helps forensic experts to know about any anomaly and its correlation with other historical anomalies.

- **Visualization And Reporting**

Visualization allows security analysts to easily understand the complex relationships in massive datasets. It provides powerful graphics and support for customized dashboards for visualizing and exploring threat data. BDSA solutions help users to perform robust search and visualization for easy threat hunting.

- **Cyber Threat Hunting**

Threat hunting is an iterative process which proactively looks for sophisticated security threats in massive datasets. It helps to detect those threats which escape existing threat detection tools. The outcome of each iteration is stored and used to further enrich the detection systems.

Table 2.2: Difference between traditional SIEM and BDSA.

Type	Traditional SIEM	BDSA
Scalability	Built upon RDBMS or proprietary flat file databases leading to non-scalable and inflexible architecture.	Highly scalable and flexible as these are built upon big data architecture.
Data model	Closed data model.	Open data model, which permits data sharing with external applications.
Visibility	Limited visibility and incomplete risk analysis as collected data cover only a small fraction of any relevant activity.	An extensive collection of data from internal and external sources provides a better understanding of threat landscape.
Context enrichment	Limited context enrichment of threat data resulting in poor threat detection and response.	Real-time context enrichment of threat data at the time of ingestion leads to accelerated threat investigation and response.
Protection	Use rule-based logic and correlation for known threats.	Advanced machine learning algorithms and automatic adaption to detect unknown sophisticated threats and frauds.
Correlation rules	Difficult to create and manage complex correlation rules.	Easy plug and play deployment help analysts to focus on investigating/remediating a potential threat.
Data complexity	The solutions are mainly focused on IT infrastructure information leading to basic security monitoring and reporting.	Incorporates large data from diverse sources to detect and prioritize highly sophisticated threats.
Search and threat hunting	Limited threat hunting and investigation due to inability to store and scale related threat data	Robust search and visualization for easy threat hunting.
Performance	Poor performance in time-sensitive investigations.	High performance and accelerated investigations.
Cost	Very high	Very low
Deployment Time	Long as complete deployment may take around 12 months.	Very short. Up and running in a few hours.

2.4.2 Use Cases

In the area of cybersecurity, a staggering amount of information is captured and analyzed for threats. Cybersecurity solutions are required virtually in every domain like finance and insurance, healthcare, retail & manufacturing, social media, media and entertainment, banking, transportation, education, energy, telecommunication, science & research, astronomy, and government agencies, defence, etc. With the advancement of BDSA, there are many frameworks and tools which can analyze large amount of information and provide the intelligent information to the customers within seconds. Some of the use cases about BDSA are discussed as follows:

- **Social Network Analytics**

Marketing and business departments are making use of BDA tools to do the behavior analysis of customers from their experiences shared on social media. The same technology can be used to monitor data shared on social networks for detecting criminal activities.

- **Network Data Analysis**

Organizations use monitoring and event correlation systems for detecting attacks by collecting and normalizing log data as well as alerts from different data sources on a network like IPS, routers, firewalls etc. The events from such integrated data are correlated to reduce false positives. A supplementary intelligence feed can be incorporated into the detection system and external factors can be correlated further to improve threat detection and prediction.

- **Network Security And Performance**

For the improvement of the security and performance of network architecture, a system can be used to analyze network traffic in real-time which can identify the abnormal traffic behavior patterns and raise alerts on the same.

- **Protection of Sensitive Data**

The solutions for protecting data monitor network traffic at strategic points like e-mail servers and gateways to identify the transmission of sensitive data at the border of the corporate network. A CTI feed can help in protecting sensitive business data by integrating it with the behavioral or keyword driven analytics and monitoring systems to identify the threats which can otherwise be a cause for loss of sensitive information.

- **Vulnerability Analysis**

The external threat intelligence feeds providing information about the new vulnerabilities (being exploited by attackers) can be integrated with the abnormal behavior pattern monitoring system to identify continuing advanced cyber-attacks on the network.

- **Detecting Insider Threats**

In order to address the problem of insider threats, the capabilities such as context based monitoring, data mining, link analysis and automated correlation, etc. can be used to generate intelligence related to insider threats. It does not only help to identify insiders with high risk identity but also the high risk activities performed by them. It is performed by analyzing a diverse set of users, applications, physical accesses and security related events for identifying abnormal behavior associated with sabotage, data theft or fraud.

- **Detection of Targeted Attacks**

BDSA can be applied to detect and predict targeted attacks like watering hole and spear phishing attacks. Potentially compromised hosts along with the URLs they have visited can be identified in large data consisting of URL access by various users in an enterprise. It can be accomplished by analyzing the frequency of access to a specific URL in a given time-period. Further, a CTI feed can proactively uncover such attacks. It helps in preventing and mitigating these attacks in future.

- **Data Leakage Detection**

BDSA can be used for detecting new data leakage scenarios, i.e., other than those identified by standard solutions for preventing data losses. For instance, the most common method for uploading the files (as a part of business requirements) could be HTTP POST. However, these could also be uploaded by malware or an insider threat for data exfiltration purpose. The application of security analytics on the data collected from diverse sources likes e-mail, file uploads can be used for detecting such data exfiltration.

2.4.3 High-Level Conceptual Architecture

Figure 2.4 depicts a high-level conceptual architecture of BDSA. The scalable architecture is capable to store, process and analyze terabytes of data collected from diverse sources in order to detect vulnerabilities and provide the appropriate response in real-time. The primary components of the architecture are discussed below:

- **Data Collection And Ingestion**

The data from diverse courses is collected and the decoders are used to load the data into big data storage tools. The data is typically stored in HDFS or Amazon Simple Storage Service (S3). The distributed data indexing can be performed by using Elasticsearch or Apache Solr.

- **Advanced Data Analytics**

The streaming as well as historical data can be processed by using big data frameworks like Apache Hadoop, Apache Spark, Apache Storm, Apache Flink, etc. to find anomalies in near real-time. The analysis of past events helps to identify better correlations with current events. Apache Hive or Spark SQL can be used for easy data access. Behavior analytics and sophisticated machine learning algorithms from libraries like Apache Mahout, MLlib, Apache SAMOA, FlinkML, etc. can be deployed to correctly identify the vulnerabilities and security risk intelligence. In addition, data enrichment can help a security organization to identify most likely threats and also provide flexible analytics. The deployment of big data tools ensures that the threat investigation time reduces from hours to minutes.

- **Interactive User Interface**

The security applications can be accessed through easy-to-use user interface and dashboard. The key applications include incident response, reporting and alerting, easy search and filtering enabling proactive threat hunting, threat prioritization/management, threats sharing, forensic analysis, statistical analysis, and cloud services. The solution can be easily deployed in local or cloud or hybrid mode.

2.4.4 Examples from Literature

A large number of incidents logged inside the SIEM system poses major challenges in analyzing and detecting security related incidents. In a case study [86], Zions Bancorporation pointed out that Hadoop and business intelligence tools extract the intelligent information more quickly than traditional SIEM tools. This section discusses the examples from the literature which makes use of BDA in the domain of security.

- **Security Event Analytics**

The organizations are collecting a massive amount of data related to software, network and people activity events for governing compliance and forensic analysis. Such massive

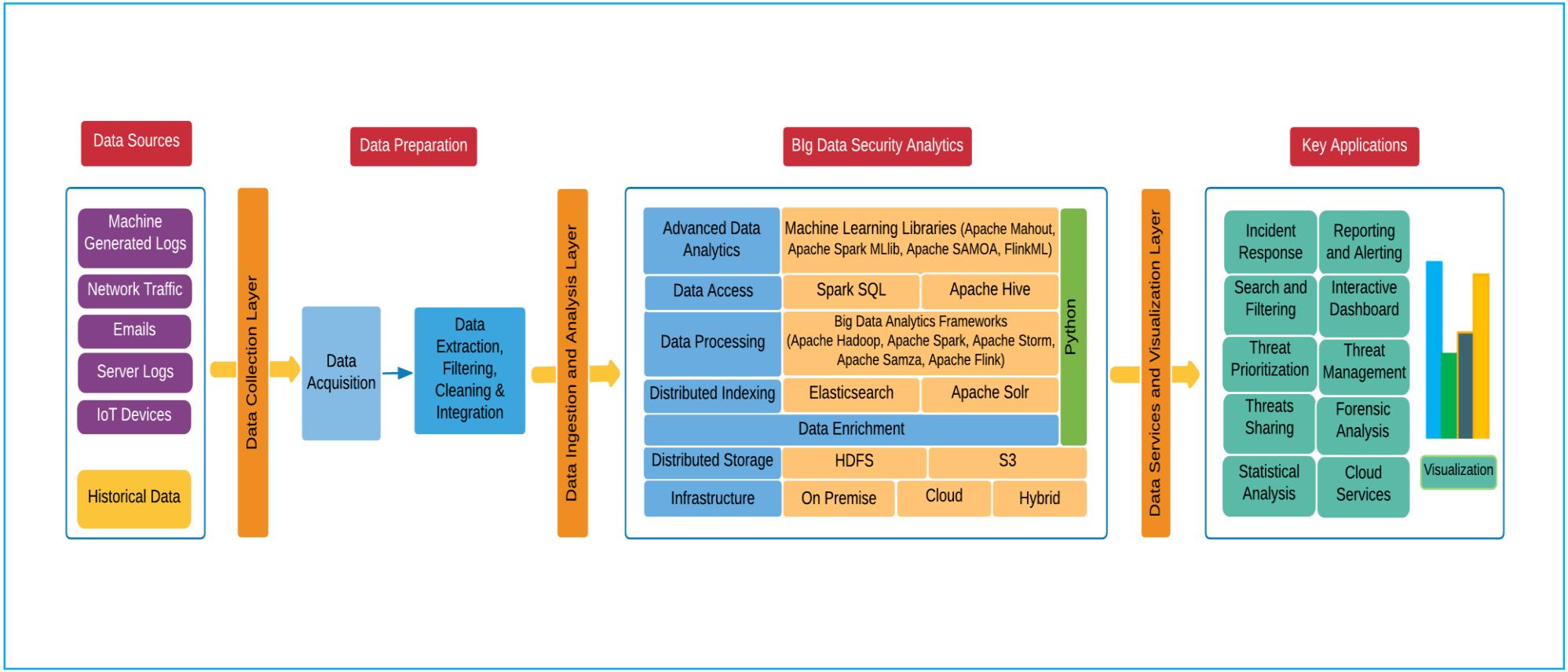


Figure 2.4: A high-level conceptual architecture of big data security analytics.

data overwhelms the existing traditional analytical solutions. The goal of BDSA fits well in this situation because more data may lead to better decision making. An accurate actionable information can be generated in real-time with the implementation of machine learning algorithms which in turn can help to deal with the future attacks. For example, a billion of the DNS events (requests and responses) collected can be analyzed from the terabytes of data to identify malicious domains.

Manadhata et al. at HP lab [87] presented a large scale graph inference approach to detect the malicious domains and the hosts which are infected with malware. They constructed a graph from the event dataset of hosts and the domains visited by them (of their enterprise). A belief propagation was used to compute the malicious hosts and domains. The approach was tested on a dataset including 2 billion HTTP requests, 1 billion DNS requests from an ISP and 35 billion network intrusion detection system alerts collected from different enterprises. This approach yields good results.

The researchers at HP lab [88] collected and analyzed terabytes of DNS events for identifying malicious domains, botnets and malevolent activities. They used several features including time to live (TTL) values for a DNS response, time stamp etc. along with decision tree and support vector machine (SVM).

- **Botnet Detection**

About 98.6 million bots were observed in 2016, registering an increase of 7% from 2015 [63]. A large amount of data collected from Netflow poses a number of challenges if processed or analyzed using traditional security tools. A research project BotCloud [89] used the big data platform for analyzing a large amount of Netflow data to recognize the infected hosts taking part in a botnet. It relies on Bottrack [90] which observes the associations between different host machines using a combination of dependency graph (created using Netflow data representing each host as a node), PageRank (for finding patterns in the graph) and DBScan clustering (for grouping the hosts having same patterns). PageRank is implemented in MapReduce as it is the most resource consuming component.

Singh et al. [91] proposed a quasi-real-time intrusion detection system build on BDA platform for detecting peer-to-peer botnet attacks using random forest (RF) algorithm. They built a distributed framework using Hive to extract dynamic network features and detected peer-to-peer botnets in near real-time by using RF algorithm.

- **Detecting APTs/Malware**

An APT (advanced persistent threat) is dreadful threat that the organizations are facing today. It is launched by the highly skilled attackers, who aim to gain access to the sensitive data of an organization for financial gains and reside there undetected for months or years. The existing traditional defense solutions for detecting anomalies are usually ineffective for detecting targeted and persistent threats. Such solutions rely on identifying outliers and involve human expertise for a closer/manual analysis to create signatures, which is not feasible with increasing volume of data as it is a time consuming process and lacks in scalability. BDA is an appropriate method for detecting APTs [92]. In [93], the authors proposed a framework using MapReduce for grouping the related security events by using the correlation analysis. They employed various algorithms along with the rules based on signature, anomaly and policy for detecting malicious activities. The data collected within the organization through web proxies, VPN and firewalls can be used for detecting APTs using BDSA.

Yen et al. at RSA lab [94] proposed an APT detection system named as Beehive, which can process vast log data from different sources in an organization for identifying malware infections and policy violation. Beehive can process nearly a billion log messages in an hour and has the ability to identify policy violations and malware intrusions.

2.4.5 Existing Solutions

With the advent of big data frameworks, the security experts can look forward to generate data-driven insights to protect their resources from any threat. Such big data security platforms can be used to detect, investigate and respond to cybersecurity threats, in near real-time. Fast detection and investigation of cyber threats lead to a quick response, which in turn reduces the impact of cyber-attacks. Modern solutions have advanced detection capabilities build upon AI, machine learning, behavior analytics, etc. These security analytics can reduce investigation time from weeks to minutes. Very recently, two open source cybersecurity frameworks – Apache Metron and Apache Spot, have become top-level projects of ASF. In proprietary software, IBM QRadar, Splunk, LogRhythm, RSA, Securonix, and E8 Security are among leading security analytics platforms [95]. These frameworks are discussed below and compared in Table 2.3:

2.4.5.1 Apache Metron

Apache Metron (www.metron.apache.org) is a real-time, scalable, advanced analytics framework for security analysis and monitoring built on the top of open source big data technologies. The streaming application makes faster detection and response to cyber anomalies. Apache Metron can store, process and analyze diverse data feeds at large scale. It uses a single repository to store security telemetry data and ingests most of such data at source using Apache Nifi. The stored data is secured via Apache Ranger using a set of access policies and can be accessed through a variety of APIs. For data visualization, Apache Zeppelin and Jupyter notebooks can be used. The key capabilities of Apache Metron are security data lake, pluggable framework, security application, and threat intelligence platform.

2.4.5.2 Apache Spot

Apache Spot (www.spot.incubator.apache.org) is an open source community driven cybersecurity analytics framework. It expedites threat detection, investigation and remediation by using BDA and machine learning. It uses Apache Hadoop data storage, and Apache Spark for machine learning and near real-time anomaly detection. Besides, Apache Spot uses context enrichment, noise filtering, whitelisting and heuristics to shortlist most likely security threats patterns.

The key features of Apache Spot are identification of suspicious DNS packets; fast threat incident and response; recognizing suspicious connects within the network resulting to most likely threat pattern; open data models providing a standard format making it easier to integrate cross application data; and collaborations allowing quick sharing of new threats among various organizations. Apache Spot can be used for the fast development of cybersecurity solutions. Some of the use cases are network traffic analytics; reduction of mean time to incident detection & resolutions (MTTR) and impacts of a breach; threat hunting using adhoc searches and queries; and cybersecurity data management offering immense economic value due to the low cost of data storage and processing with Apache Hadoop.

2.4.5.3 IBM QRadar

IBM QRadar [96] is a security analytics platform used to monitor, analyze and store massive amount of data in real-time, converting raw security data into useful patterns to

detect malicious activity. It can quickly identify behavioral changes deviating from regular patterns, network traffic anomalies or other user defined activity exceeding a threshold. QRadar can efficiently perform rapid searches and scale up for speed and data storage capacity using automatic load balancing technology. QRadar includes cognitive security capabilities from Watson and security automation from Resilient Systems. It offers flexible deployment options like on-premises, cloud or as a managed service. Some of its key functionalities are vulnerability and risk management, forensic analysis, deep packet inspection, user behavior analytics, and incident response.

2.4.5.4 Splunk Enterprise Security (ES)

Splunk ES [97] is a flexible yet powerful security framework build upon Splunk operational intelligence platform. Splunk offers over 25 commonly used machine learning commands which can be applied to large data for detection, alerts and analysis [98]. To create custom solutions, Splunk provides a machine learning toolkit allowing access to additional commands and open source algorithms. It enables users to capture, monitor, and visualize big data from diverse sources like security devices, systems and applications. Splunk ES uses advanced analytics for threat detection and intelligence. The key features of Splunk ES are better security posture/operations, identification of internal and advanced threats, threats prioritization and management, and better informed decisions.

2.4.5.5 LogRhythm

LogRhythm [99] platform has high performance, distributed, and component based scalable architecture to provide real-time actionable intelligence. Its main components are data collector, system monitor, network monitor, data processor, data indexer, AI engine, and platform manager. It uses advanced analytical techniques like machine learning, behavior profiling, statistical analysis, sophisticated correlations, blacklisting, and whitelisting for automatically detecting advanced threats. It also uses Elasticsearch based indexing along with its patented data processing technology to provide scalable machine data analytics. It offers easy to develop customized analytics using GUI based rules. The platform deployment is available on-premises using LogRythm appliances, customer hardware, virtual infrastructure, and private cloud. The feature list includes advanced analytics, file integrity monitoring, endpoint threat detection, identity analytics, network threat detection, precision search, and user and entity behavior analytics (UEBA).

2.4.5.6 RSA Security Analytics

RSA NetWitness Suite [100] (formerly RSA Security Analytics) is a tool for analysis of massive security data from multiple sources in real-time. It uses advanced analytics to detect threats and discover cyber-attacks which elude log-centric SIEM and signature-based tools. The key benefits include high performance and scalability, automated behavior analytics to identify advanced threats, threat prioritization, recreation of full sessions (browsing, email, FTP, etc.) during threat investigation, etc. The solution can be deployed on premise, in the cloud (AWS and Azure) or using a hybrid approach. The key features are data processing from multiple sources, real-time and historical analysis, statistical analysis and machine learning, turnkey capabilities, intelligent prioritization, and advanced threat detection.

2.4.5.7 SNYPR

Securonix SNYPR [101] is a big data security analytics framework built on Apache Hadoop which extracts useful threat intelligence from a large collection of data from diverse sources. This distributed solution scales up horizontally to ingest and analyze a massive amount of data by adding more Hadoop nodes. It uses UEBA, sophisticated machine learning algorithms and behavior analytics to enable predictive threat detection and security risk intelligence, in real-time. The behavior based techniques detect any variation in normal patterns and remove the noise associated with incremental changes in user behavior. The open data model allows organizations to customize their solution without creating separate data stores making it cost effective. The key features include insider threat detection, actionable intelligence with behavior analytics, intellectual property protection, data breaches prevention, interactive forensic investigation, and visualization dashboard.

2.4.5.8 E8 Security

E8 Security Fusion Platform (www.e8security.com) is a scalable security analytics platform. The underlying Apache Hadoop infrastructure allows it to efficiently manage, store and use terabytes of data from diverse sources supporting thousands of users and endpoints. The behavioral intelligence platform identifies any risk to data breach and recognizes early warning signs for any threat intrusion. It uses advanced analytics for fast threat detection of any suspicious activity and reduces the threat investigation time from

Table 2.3: Comparison of big data security analytics frameworks.

Comparison Metrics	**Apache Metron**	**Apache Spot**	**IBM QRadar**	**Splunk ES**	**LogRhythm**	**RSA NS**	**SNYPR**	**E8 Security**
Created in	September, 2014	February, 2016	October, 2012	April, 2014	--	December, 2015	2011	2013
Open Source	✓	✓	×	×	×	×	×	×
Most Stable Release (as of January, 2018)	Apache Metron 0.4.2	Apache Spot 1.0	IBM Security QRadar 7.3.1	Splunk Enterprise Security 5.0.0	LogRhythm Enterprise 7.2	RSA NetWitness Suite 10.6.5	SNYPR 6.0	E8 Security Fusion Platform 2.2
Data Processing Model	Streaming	Batch, Streaming	Batch, Streaming	Batch, Streaming	Batch, Streaming	Batch, Streaming	Batch, Streaming	Batch, Streaming
Platform	Apache Spark, Apache Storm	Apache Hadoop, Apache Spark	IBM Watson	Inbuilt, Apache Hadoop	--	Apache Hadoop	Cloudera Hadoop Distribution	Apache Hadoop, Apache Spark
Data Store	HDFS, HBase	HDFS	Inbuilt	Inbuilt	--	Data Warehouse	HDFS, HBase	HDFS
Cluster Management	Apache Ambari	Spark (YARN)	High-Availability Management	Inbuilt	High-Availability Management	High-Availability Management	Cloudera Manager	Cloudera Manager
Product Maturity	Medium	Medium	High	High	High	High	Medium	Medium
Advanced Analytics								
Machine Learning	✓	✓ (MLlib, Spot Machine Learning)	✓	✓ (Splunk Machine Learning Toolkit)	✓	✓	✓	✓

AI	✓	✓	✓ (IBM Watson AI)	✓	✓	✓	✓	✓
Behavioral Analytics	✓	✓	✓	✓	✓	✓	✓	✓
Deployment Mode								
On-Premise	✓	✓	✓	✓	✓	✓	✓	✓
Public/Private Cloud Support	✓ (Amazon EC2)	✓	✓	✓	✓	✓ (Amazon EC2 and Azure)	✓	×
Hybrid	✓	✓	✓	✓	✓	✓	✓	×
Key Capabilities								
Threat Intelligence	High	High	High (IBM X-Force)	Medium	High	Very High	High	Low
Real-time Anomaly Detection	High	Very High	Very High	Very High	Medium	Medium	Medium	Medium
Threat Prioritization	✓	✓	✓	✓	✓	✓	✓	✓
Scalability	Very High	Very High	Very High	Very High	Medium	Very High	Medium	Low
Security Automation	High	High	High	High	High	Medium	Very Low	Low
Visualization	✓ (Zeppelin, Jupyter)	✓ (Spot UI, Jupyter)	✓ (IBM QRadar Pulse)	✓	✓	✓	✓	✓

-- Information Not Available

hours to minutes. It analyzes present as well as past data to find insights about past events and identifies the correlation with current events. The easy search and filtering facilities make it possible to proactively hunt threat indicators. The key features include advanced detection, retrospective analysis, search and filter, GUI based analytics, and unsupervised machine learning.

2.5 Malware Classification: A Case Study

Undoubtedly, malware is one of the most critical and challenging security threats in the Internet world. Security organizations have to handle tens of thousands of malicious samples every day. On a daily basis, over a million malicious files are submitted to public malware sample repositories for their identification. The adoption of mobile devices and cloud computing services have provided opportunities to attackers for discovering vulnerabilities and launching the malware attacks. In recent times, the digital devices have seen tremendous growth in their volume with the advent of IoT and have begun to face more malware attacks. VirusTotal (www.virustotal.com), a subsidiary of Google, analyzes files and URLs for identification of malicious files. The weekly statistics of VirusTotal shows that on an average, over a million of malware samples are submitted per day. In the year 2015, McAfee labs reported more than 433 million samples, with 12% growth in a second quarter and in the same time interval, there was 17% growth in the malware samples in mobile devices, which is quite significant considering the growing number of mobile devices [102].

In 2016, malware menace continued on the threat landscape. There was 36% rise in ransomware attacks worldwide and about 98.6 million bots were observed, registering an increase of 7% from 2015 [63]. According to a recent threat report of Quick Heal [103], around 973 million malware (targeting Windows systems) were detected in the year 2018, meaning nearly 2,666,957 per day, 111,123 per hour, 1,852 per minute. In addition to huge volume, the malware are becoming more sophisticated and complex as well. The WannaCry ransomware exploited vulnerability in Windows server message block (SMB) protocol and spread so quickly that it covered 150 countries within 24 hours of its outbreak [104]. The attack infected around 400,000 machines worldwide and 98% of these were running Windows 7 as operating system. The attack resulted in major outage in healthcare, transport, banking, financial institutions, and telecom etc. These staggering statistical figures show a huge growth of malware data. In such scenarios, malicious

samples are bound to grow at an astonishing rate and thus, there is a need to develop a system, which can do analysis of static and dynamic attributes of malicious files at large scale and provides the insights to the end-user in no time.

The malicious binaries are showing tremendous growth in volume (increasing cyber threat landscape), variety (adopting advanced malicious approaches) and velocity (cyber threat uncertainty) [105]. The huge volume, variety, and velocity of malware data have become overwhelming for traditional malware analysis technologies and require new methods for managing threats. The malware data has now become a big data problem and there is a need to adopt big data technologies, which can analyze and process information at large scale.

A malware is a piece of software program that accomplishes the destructive intent of an attacker [106]. According to National Institute of Standards and Technology (NIST), "malware is a program that has the intent of compromising the confidentiality, integrity, or availability of the victim machine and its resources" [107]. There are different classes of malware depending upon their characteristics like the ways they infect the systems, get propagated etc. The most common classes are Virus, Worm, Trojan horse, Spyware, Backdoor, Botnets, Rootkits and Adware etc. [108]. These classes may overlap most of the time as these usually share the characteristics of diverse classes.

2.5.1 Overview of Malware Analysis

Malware analysis is performed to gather comprehensive information about malware functionalities so that it can be neutralized or eliminated from the system. In general, malware analysis aims to handle or prevent any network intrusion. With a large number of malicious programs being encountered every day, performing malware analysis is too critical to maintain computer security. There are two fundamental approaches to conduct malware analysis: static malware analysis and dynamic malware analysis. The static malware analysis understands the functionality of malicious code without running it; while dynamic malware analysis runs the malicious code in a safe environment. The static malware analysis is usually the first step to study a malicious file and can be further divided into two parts: basic static analysis and advanced static analysis. The basic static analysis only checks the malicious file, ignoring the instructions inside the malicious code and thus, provides trivial information about its functionality. The advanced static analysis makes extensive use of reverse engineering and loads the malicious code into a

disassembler to understand its behavior. On other side, the basic dynamic analysis runs the malicious code safely in an environment and extracts the detailed information about its activities. The advanced dynamic analysis uses a debugger to know more about the internal state of a running malicious code [109]. Figure 2.5 represents different types of malware analysis.

The main advantage of static techniques is to explore all possible execution paths of a malicious file, while dynamic analysis is limited to a single execution path. Another benefit is that the implementations of static methods are fast as compared to dynamic techniques and thus, static analysis is preferred when speed is critical [110]. When a malicious file is packed, it should be unpacked before performing any static analysis. The malware authors keep on evolving sophisticated and complex evasion techniques to avoid their detection by static or dynamic analysis and thus, security experts are forced to adopt a hybrid approach which integrates both static and dynamic analysis techniques.

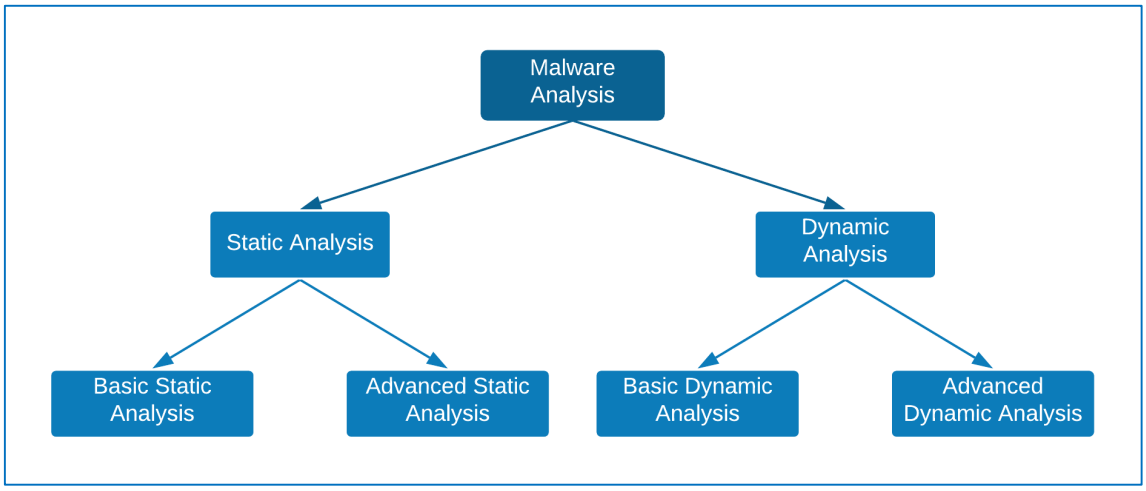


Figure 2.5: Types of malware analysis.

2.5.2 Malware Detection Methods

The anti-virus software programs have been around since early 1990s. Subsequently, with the proliferation of mobile devices, cloud computing and expansion of Internet access, malware infection increased and so did the anti-virus industry. The commonly used methods for malware detection are based on signature, heuristic, behavior, and machine learning. Mostly, the anti-malware tools make use of signature-based methods for identifying malware samples, which are inadequate to protect against zero-day attacks. Moreover, the exponential growth of malicious binaries has already made malware

detection a big data problem which overwhelms the traditional solutions. Nowadays, the machine learning based methods is a popular choice to classify the behavior of a file on the basis of features collected using static or dynamic analysis of the file.

2.5.3 Malware Classification using Machine Learning Methods

This section discusses the state-of-the-art carried out in classifying malware binaries or identifying those that do not fall under existing classes and thus deserve for the closer human analysis. Over the last one and half decade, researchers have proposed various approaches which use malware features (static or dynamic or integration of both) obtained after performing malware analysis along with machine learning methods for detecting and classifying malware binaries.

Schultz et al. [111] introduced the concept of machine learning and data mining for identification of malicious files. They made use of static features, namely, byte sequence, portable executable and strings along with machine learning algorithms for malware classification. Later on, Kolter et al. [112] used n-gram to improve the results for malware detection. After that, many techniques [113-117] were proposed and developed for detecting malware using static malware features.

Through their work, the authors [118] concluded that features obtained through static analysis are not adequate to identify malware correctly as this type of analysis is not able to detect the packed and obfuscated malware. They advised that features obtained through dynamic malware analysis is an essential complement to static analysis.

Several researchers use dynamic malware features like sequence of API calls [119], network behavior [120] sequence of system calls [121, 122] for classifying malware. In [123-127], the authors performed experiments to show that a single type of feature either static or dynamic is insufficient for precisely classifying malware samples. Thus researchers have adopted the integration of static and dynamic features for developing improved systems for identification of malware and classification. All these techniques for malware detection are not scalable and are evaluated on a small dataset.

Recently, researchers have started to explore the potential of big data in network security like log analysis, malware detection, network traffic monitoring systems. In [128], the authors proposed a security model which considers the security analytics using big data.

They pointed out that the problem of malware detection as a big data problem as anti-virus vendors are facing a large flux of malware samples every day.

Singh et al. [91] proposed a distributed framework for extracting dynamic network features. They used decision tree model for detecting peer-to-peer botnet in quasi real time. For developing their framework, they used Hadoop, Hive and Mahoot. The decision trees are deployed applied for data analysis as these involves a rich set of rules which can be understood easily [129, 130].

Panigrahi et al. [131] pointed out that cloud service provider needs to scan the database content to prevent from malware. They proposed a model for quick identification of malware. They implemented the existing pattern matching algorithms in map-reduce model for matching signature of malware. The drawback of the proposed system is that it is only capable of detecting the existing ones but not the unknown malware.

Miller [132] proposed and implemented a framework which uses Hadoop Distributed File System (HDFS) and Apache Spark. The proposed framework is evaluated on a use case of malware samples containing more than 1 million samples. The experimentation work demonstrates that the detection outcome can be improved by allowing review of a fraction of malware samples by human.

Literature provides the idea that the advantages of various techniques could be combined into a single classifier to improve the performance. Thus various ensemble learning methods have been designed to deal with the problem of detecting unknown malware with more accuracy. Ensemble learning unites a set of base or weak classifiers to obtain a better compound classifier. The subsequent part of the section explores various contributions made by the researchers and practitioners in the field of malware detection and classification using ensemble learning.

Zhang et al. [133] presented a method for classification of malware based on n-gram feature. Afterwards, information gain method is used for selected the best n-gram features. Then, probabilistic neural network (PNN) is used to build the classifiers followed by Dempster-Shafer theory to combine the individual predictions made by PNN classifiers. They evaluated the proposed method on a dataset consisting of 450 malicious and 423 benign files. The results show better receiver operating characteristic (ROC) curve for the ensemble of PNNs.

Mukkamalaa et al. [134] made use of majority voting method to ensemble the prediction of various classifiers for detecting intrusions in the network traffic. They used the base classifiers as SVM, multivariate adaptive regression splines (MARS), and three types of artificial neural networks (ANNs). A dataset of DARPA is used for evaluation purpose. They showed that majority voting method improves the accuracy of the detection.

Menahem et al. [135] used three categories of features, i.e., n-gram, function-based and portable executable (PE) features. They constructed five different datasets and considered five base classifiers, OneR, VFI, K-Nearest Neighbour (KNN), Naïve Bayes (NB), and C4.5 decision tree to process these datasets. Afterwards, majority voting, distribution summation, Bayesian combination, performance weighting, stacking and troika were used to combine the base classifiers. The experimental results showed that troika and stacking outperformed the base classifiers. The distribution summation and Bayesian combination gave the same result as that of best base classifiers and voting, but NB and performance weighting performed poorly than base classifiers.

Ye et al. [136] used application programming interface (API) calls and strings as features. They constructed eight base classifiers on various combinations of features. Afterwards, they used simple voting method to combine the base classifiers. The results pointed out that the method outperformed the base classifiers.

Guo et al. [137] proposed a method in which they categorize the API calls into seven classes and used these to construct base models. The predictions made by the base models are combined and the results are compared with best base classifiers and voting method.

Landage et al. [138] used opcode sequence of the malware samples to build base classifiers. The output of the three base classifiers was combined using majority voting and veto-based voting. The experimental results show that veto-based voting method gave better detection rate as compared to majority voting method, but increased the false positive rate (FPR).

Sheen et al. [139] constructed a set of heterogeneous base classifiers on the basis of the API calls and features extracted from PE header. They proposed two ensemble methods which were used to select and combine a set of base classifiers. The method attained 99.7% detection rate which was better than the traditional methods of bagging, boosting and stacking.

A similar approach has been proposed in [140] where the authors divided the features into various subspaces and each of these is used to construct the base classifiers. Afterwards, they used an evolutionary algorithm to assign weight to each base classifier. A weighted voting method is used to combine the selected base classifiers. The experimental results show that this method provides better results.

Ozdemir et al. [141] proposed a method for detection of Android malware on the basis of API calls. Using these API calls, they build different base classifiers. Then the predictions made by base classifiers are combined using majority voting. Their results show that the proposed method has improved the accuracy in detecting malware.

Another method [142] is proposed for the detection of Android malware in which the authors used permissions and API calls. These features were used to construct six base classifiers. Then a collaborative decision fusion method was used to combine the predictions of all classifiers. The authors claim that better results were obtained using this method as compared to the traditional methods like Adaboost and bagging.

Yerima et al. [143] proposed a technique 'Droidfusion' for combining the predictions of base classifiers. It is based on multilevel architecture. They proposed four ranking algorithms to rank the base classifiers and then considered their combinations to combine the results of base classifiers. They used four different datasets to show that their method is better as compared to other traditional ensemble approaches.

Bai et al. [144] designed two ensemble methods along with the integrated feature set based on opcode and n-gram. They used two datasets to evaluate the performance of malware detection of the proposed schemes. The results show that the proposed methods have improved the generalization performance of the learning models.

In [145], the authors demonstrated that combining a set of base classifiers for constructing an ensemble instead of considering all improves the generalization performance. They used greedy method to add iteratively new classifiers that maximize the performance of ensemble according to area under curve (AUC). In each iteration, a number of base classifiers are randomly selected and the performance of the current ensemble is evaluated. The classifier leading to the best generalization performance is selected as a part of ensemble. This process is repeated until a maximum ensemble size is reached.

Li [146] pointed out that there should be diversity in the classifiers in order to make the ensemble learning more effective. In ensemble methods using multiple classifiers, diversity is obtained by making use of different types of classifiers each containing an explicit or implicit bias. Combining such type of classifiers leads to achieve accuracy that none of the individual classifiers would be able to achieve.

2.5.4 Malware Classification using Deep Learning Methods

Many state-of-art machine learning based methods suffer from high FPR and low scalability restricting their wider adoption and deployment. In recent years, deep learning, a subfield of machine learning, has resurged and reported outstanding performance in tackling many classification problems in a wide range of fields.

Kephart [147] was the first to use a shallow NN for malware classification. Dahl et al. [148] investigated deep learning for malware classification for dynamic analysis achieving error rate of 0.49 and 0.42% for a single and an ensemble NN respectively. David et al. [149] used deep belief network (DBN) having capability to generate malware signature automatically and using these signatures to classify malware binaries. Pascanu et al. [150] explored recurrent neural networks (RNNs) to model the sequential behavior of malicious binaries. The authors applied long short term memory (LSTM) and gated recurrent units to achieve good classification performance. Hou et al. [151] collected the system calls from Linux kernel and constructed the weighted directed graphs. They applied deep learning on the graph based features for detecting Android malware. Saxe et al. [152] proposed another deep learning based method for malware detection which uses two-dimensional binary program features. Kolosnjaji et al. [153] presented a method to detect and predict malware family based on the system call sequence using DNN architecture and reports better results than Hidden Markov Models and SVMs. Huang et al. [154] proposed a solution for dynamic malware binary classification to report modest improvement over shallow neural architecture. Recently, Sun et al. [155] proposed DeepMalware, a neural network architecture, using small dataset and used convolutional layers, inception design, and bidirectional RNNs to improve the performance of malware detection. Ye et al. [156] studied the design of deep learning architecture using Windows API calls extracted from PE files for malware detection. They proposed a heterogeneous deep learning framework comprising of an auto-encoder stacked up with multilayer restricted Boltzmann machines and a layer of associative memory for detecting zero-day

malware. Wu et al. [157] proposed a reinforcement learning (RL) based model to automatically generate malware samples which can evade traditional machine learning based detection systems. Through self-learning, the model helps in improving the malware detection rate. Rhode et al. [158] uses an ensemble of RNNs to determine whether a binary is a malware in first five seconds with 94% accuracy by taking a snapshot of malware behavioral data.

From the literature review, it is found that most of the existing techniques based on machine learning and deep learning proposed for detecting malicious programs are not scalable and thus unable to handle the increasing volume and diversity of malware families. Moreover, these are evaluated on a limited number of malicious and benign samples collected from a fixed set of sources.

2.6 Summary

This chapter provided the background of BDA and identified the role of BDA, machine learning and deep learning in cyber security. It also provided the use cases, examples from the literature, and a comparison of existing popular solutions of big data security analytics. It discussed the background of malware which is considered as a use case for this research work. It also provided the literature review of the techniques for malware analysis and classification using machine learning and deep learning methods.

Chapter 3: Problem Formulation

This chapter lists the research gaps that have been identified while carrying out literature review. These gaps identify the problems with the existing malware detection techniques. Besides, the chapter presents the problem statement, lists the research objectives and concludes by defining the brief methodology for achieving these objectives.

3.1 Research Gaps

Although big data analysis techniques have made much progress recently, still several challenges remain. In many applications, the data analytics is a clear bottleneck due to the complexity of data which needs to be analyzed, and the lack of scalability of the underlying algorithms. As most of the data is unstructured, so transforming such content into a structured format for later analysis is not only difficult, but is also one of the major challenges. Some of the major research gaps which are identified in literature review are given below:

1) Although various analytical and statistical modeling techniques can mine big data, these techniques do not always provide actionable insights.
2) Big data collected from distributed systems is heterogeneous in nature. Thus, the information from such heterogeneous systems needs to be correlated for analysis.
3) The analysis of big data should be interactive to have real-time answers. To scale the complex query processing techniques to the scale of big data while keeping the interactive response time is one of the open research problem.
4) Existing data warehouses and relational databases excel at processing structured data. As most of the data is unstructured and semi-structured, so new solutions (techniques and algorithms) need to be developed to handle large scale data, though a few solutions are already available.

5) In the analysis phase, the complexities pertaining to multi-tenanted clusters (i.e. several programs running concurrently) are yet to be understood properly.
6) Ensemble learning and deep learning have not been investigated much for detecting unknown malware at large scale which can help to reduce the false positive and false negative rate (FNR) in malware detection.

3.2 Problem Statement

Big data represents the data deluge which is happening at the unmeasurable rate and affecting nearly all aspects of our life [9, 10, 47, 55, 65]. The massive growth of data has overwhelmed the traditional tools and approaches, and thus there is a need to develop new tools and techniques to realize the true potential of big data [46, 48]. In fact, the data-driven discovery approach is an emerging paradigm for computing [3] and the fundamental challenge is to analyze the massive data so as to extract the novel insights which were not feasible earlier with small data [54-56, 59].

Malware is one of the most critical and challenging cyber threat to the Internet. Due to fast adoption of mobile devices, cloud computing services, and IoT, these are ceaselessly emerging in terms of the volume (increasing cyber threat landscape), velocity (cyber threat uncertainty), and variety (adopting advanced malicious approaches) [47, 102, 105]. The exponential growth of malware samples has made it a popular big data problem. Moreover, malware detection is an important problems in cyber security, not only for security organizations but also for the society as a whole. The traditional approaches based on signature or heuristic methods for malware detection are impractical in handling huge influx of malware on daily basis [66, 67]. This has opened up a series of novel challenges and opportunities for the security researchers as well as practitioners.

In response to the abovementioned problem, we intend to propose state-of-art machine learning and deep learning based solutions for malware detection at large scale. These solutions will consider an integrated set of static and dynamic features of malware, and will be developed using open source tools and libraries for big data, machine learning and deep learning.

3.3 Research Objectives

The objectives of the proposed work are given below:

1) To better understand and explore the concepts, techniques and various tools of big data processing and analysis.
2) To propose a novel framework that provides a set of solutions (techniques and algorithms) for analyzing big data to provide actionable insights to decision makers.
3) To test and validate the proposed work.

3.4 Research Methodology

We have followed a certain set of steps to accomplish the research objectives, the workflow of which is illustrated in the Figure 3.1. Thereafter, the methodology to achieve each objective is described separately.

Objective 1:

- A bibliometric study of research publications is conducted to understand the current state, evolving disciplines and research trends of big data.
- A comprehensive review of big data is carried out to understand its evolution as well as tools required to process it. The pros and cons of various open-source big data tools are identified.
- Reviewed the significance and role of big data analytics in cyber security and identified various use cases, including malware detection, pertaining to big data security analytics. In addition, the existing open-source deep learning tools are identified and discussed.
- Identified the techniques to analyze and classify malicious binaries using machine learning and deep learning.

Objectives 2:

- Prepared malware dataset by setting up an automated malware analysis environment using distributed Cuckoo sandbox with Windows 7 as a guest machine. A statistical analysis of 1,70,000 malware samples (having disk size of 2.21 TB) using Apache Spark has been performed to study their behavioral trends over a period 2010 to 2017.
- Proposed a scalable architecture for zero-day malware detection on the top of Apache Spark using its scalable machine learning library (MLlib). Different machine learning

models are implemented and compared to identify the best model for the dataset on the basis of evaluation metrics like FPR, FNR and accuracy.

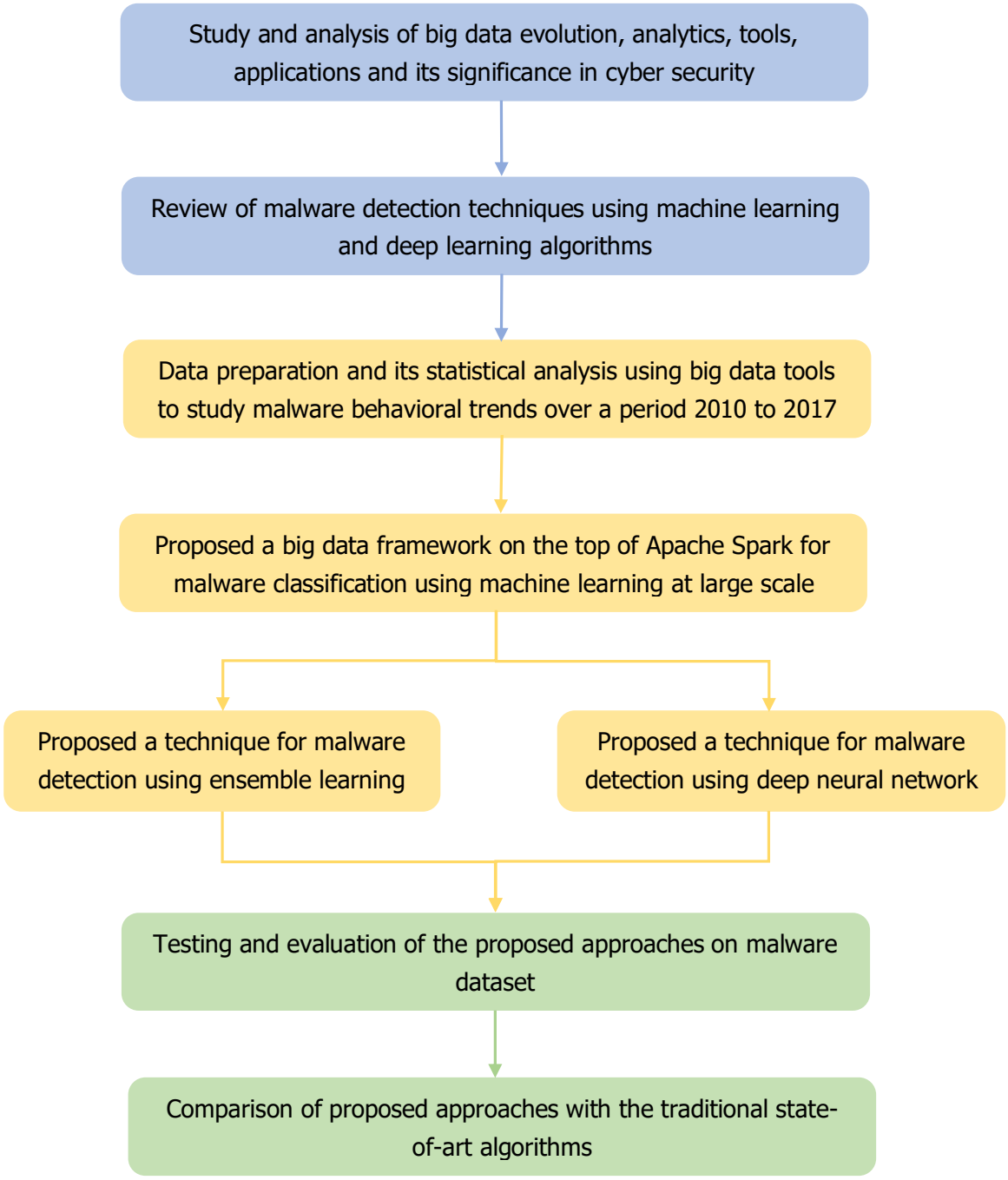


Figure 3.1: Workflow of the research methodology.

- Proposed two methods based on ensemble learning using big data architecture for improving the performance of malware detection. In first method, the weighted voting

strategy of ensemble learning is adopted, and the second method selects an optimal set of base classifiers for stacking purpose. The resulting classification model improves the generalization performance as compared to traditional ensemble methods.

- Proposed a framework for malware detection which brings together the benefits of Apache Spark – a big data processing engine, and Keras along with TensorFlow, a deep learning framework. Apache Spark is deployed for efficient preprocessing of data, and Keras along with TensorFlow is used to develop and train deep neural network. The resulting classification model gives better performance than the traditional machine learning algorithms.
- The proposed solutions are implemented on Apache Spark cluster, consisting of a master and up to four worker nodes. All nodes are having Ubuntu 16.04.2 as operating system. The hardware configuration of master node is Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD. The worker nodes are having configuration as Intel(R) Core TM i5-8250U CPU @1.6 GHz, 8 GB DDR4, and 1 TB HDD.

Objectives 3:

- The performance of the proposed models is evaluated in terms of true positive rate (TPR), FPR, FNR, precision, accuracy, and F1-score.
- The performance of the proposed solutions is compared with the traditional state-of-art machine learning algorithms.
- The scalability of the proposed solutions is confirmed by considering datasets of different size and running the model in distributed environment on varying number of worker nodes.

3.5 Summary

This chapter listed the research gaps which are identified in context to big data and malware detection using various techniques. Considering the research gaps, a problem statement was formulated which aims to propose a state-of-art solution for malware detection at a large scale. Thereafter, the chapter presented the research objectives and explained the research methodology and its workflow describing the various steps to accomplish each research objective.

Chapter 4: Architecture for Big Data Analytics

The exponentially growing data overwhelms the traditional approaches and thus, opens a window for many novel challenges as well as opportunities in terms of malware data analysis. Apart from the problem of handling malware threat at large scale, it should be understood that their detection and removal is not sufficient, but it is more critical to extensively comprehend their behavior to know the context and the goals of security breaches. Moreover, a thorough understanding of the behavioral trends of historical and current malware binaries can enable big data security experts to drive deep insights to understand threat landscape.

This chapter presents the design of the proposed architecture for understanding malware behavior trends from year 2010 to 2017. It discusses the steps to prepare malware dataset by setting up an automated malware analysis environment using distributed Cuckoo Sandbox with Windows 7 as a guest machine. Further, various open-source tools used to implement the framework have been described. The chapter also discusses the methodology and evaluation results obtained though statistical analysis of malicious binaries which can be shared with the stakeholders to focus on the prevalent areas. This chapter partly accomplishes objective 2.

4.1 introduction

As malicious code is generally very complex and difficult to understand, it is always better to focus on key features obtained after analysis. These can be used to group and classify unknown malware binaries into their existing families. Moreover, the information gathered can be used to find out the malware trends and may also help to take preventive measures to evade future attacks. Though substantial research has already been done on malware analysis and classification at small scale, very small work has been done so far on analyzing the evolving trends of malware behavior for developing an approach to deal

with malware binaries proactively. Bayer et al. [159] evaluated the malware behavior from the reports generated by Anubis malware analysis system. They discovered the malware behavioral patterns for malicious samples collected over a period of 2 years from February 2007 to December 2008. In [160], the authors have done similar work performing an automated analysis of about 0.1 million malicious specimens in a Cuckoo Sandbox for mining intelligent information regarding malware attributes and behaviors.

Daisuke et al. [161] analyzed the network activities of malware by executing these on a real machine which is connected to the virtual Internet. They carried out the analysis in two passes. The first pass was designed to respond interactively so as to capture the wide variety of network behaviors. The second pass was configured on the basis of the results obtained from the first pass so as to focus on specific behaviors. Lorenzo et al. [162] presented a method for detecting bot-infected hosts using groups of network flows created by monitoring their network traces and using hierarchical clustering algorithms. A similar work was carried out by Christian et al. [163], who analyzed more than 100,000 malware samples in their Sandnet environment to study the network activities. In their study, they focussed on two protocols, namely, DNS and HTTP.

However, the aforementioned solutions are not scalable to handle large volumes of malware data, which is too critical nowadays. Moreover, these solutions have used Windows XP as the virtual machine (VM) to execute the malicious binaries, whose support has been stopped by Microsoft since 2014 and its user base is going down. As malware data is growing continuously, malware analysis needs to use big data technologies. The proposed solution provides a scalable platform on the top of Apache Spark to analyze the attributes of malware binaries on a large scale. These attributes are generated while executing malicious samples on a virtual machine having Windows 7 as the guest machines for studying their behavioral trends. It enables end-users to draw insights from malware analysis and generates threat intelligence helping them to understand the threat landscape for taking preventive measures to avoid future threats.

4.2 Proposed Architecture

This section presents a scalable architecture for analyzing attributes of a large number of malware binaries. The framework uses Apache Spark for distributed data processing, HDFS for distributed storage, and Python language for performing data analysis. Figure

4.1 depicts the high-level components of the proposed architecture, and the subsequent subsections describe these components in detail.

4.2.1 Data Preparation

A dataset consisting of 1,70,000 malware and 98,150 benign files is prepared. The malware samples (aiming Windows 7 OS) from diverse repositories like VirusShare (www.virusshare.com), VX Heaven (www.vxheaven.org), and NoThink (www.nothink.org). These repositories are extensively used by security researchers and are easily accessible in the public domain after free registration. The downloaded malware samples are decompressed and scanned with AVG AntiVirus (www.avg.com) to confirm their maliciousness. The samples are filtered to only keep unique malware binaries (based on their MD5 hashes). In addition, the benign Windows Portable Executable (PE) files are collected from a clean Windows XP, 7, and 10 installations directory. Some of the benign samples are collected from online free software archive, such as CNET download (https://download.cnet.com/windows/) and Softonic (https://en.softonic.com/), and our local development lab.

Each sample is made to execute in a virtual environment using Cuckoo Sandbox (www.cuckoosandbox.org), a widely used open source automated malware analysis system. The timeout of 90 seconds was set for each sample and the critical timeout was set as 180 seconds. Cuckoo Sandbox is installed on a host machine with Ubuntu as OS and the Windows 7 guest machines are configured using Oracle VirtualBox (www.virtualbox.org). It is integrated with VirusTotal (www.virustotal.com) to fetch metadata information for each malware sample.

Cuckoo Sandbox produces a comprehensive analysis report for each sample in Java Script Object Notation (JSON) format after performing malware analysis. We have opted for JSON format as it is completely language independent as well as uses simple conventions familiar to most of programming languages including Python. This format is commonly used and is easy to read as well as parse. The set of JSON reports so produced form a raw dataset to proposed big data platform for further analysis. The JSON reports contain various static (file metadata, file size, packer detection, and sections' information) and dynamic attributes (like DLLs, dropped files, Windows API calls, mutex operations, file activities, registry activities, network activities, and process activities) of malware binaries and form a raw dataset to the proposed architecture for further analysis.

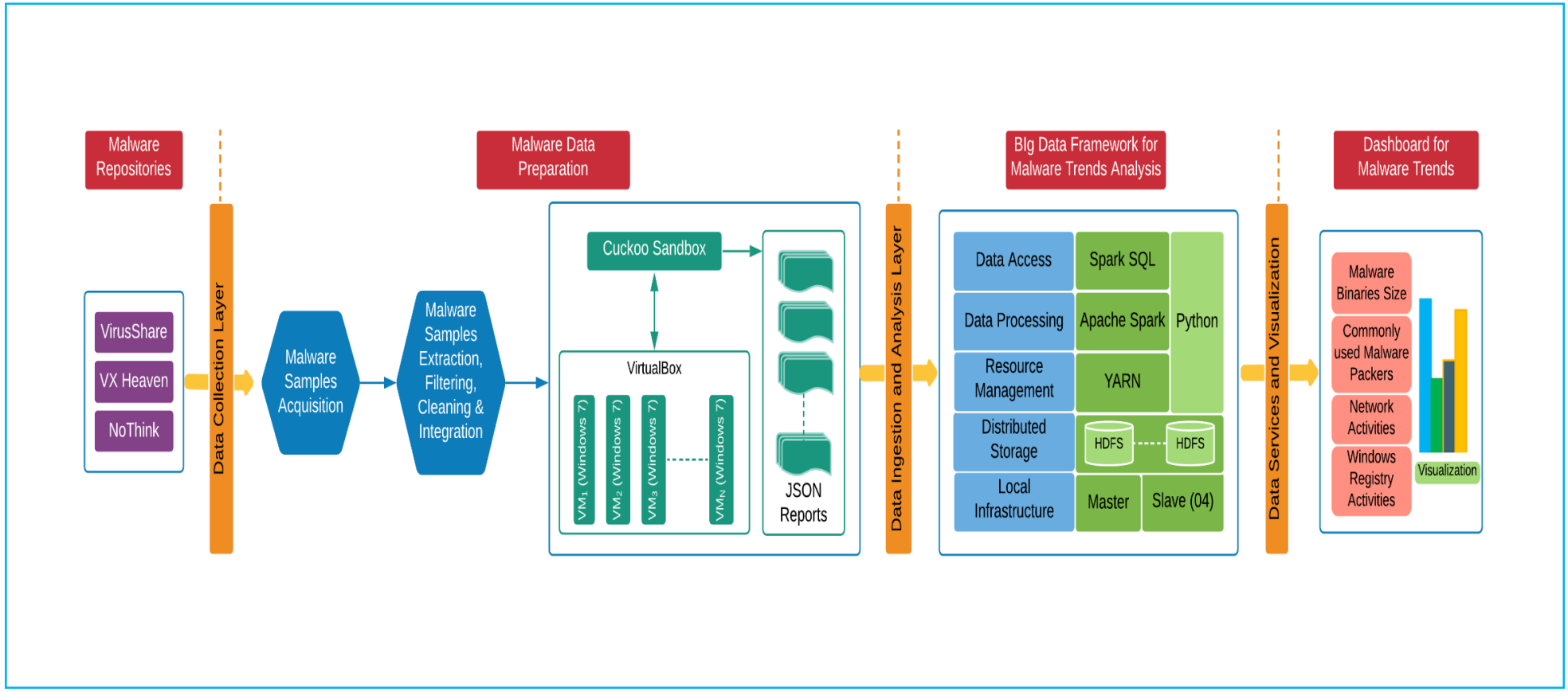


Figure 4.1: Block diagram of proposed architecture for malware trends analysis at large scale.

Cuckoo Sandbox is integrated with VirusTotal which provides metadata information for each malware sample. The 'first seen field', i.e., the date on which a malware sample is submitted for the first time to VirusTotal, is considered as the date of discovery of malware sample. Figure 4.2 shows the number of malware samples in the dataset grouped by year of their discovery.

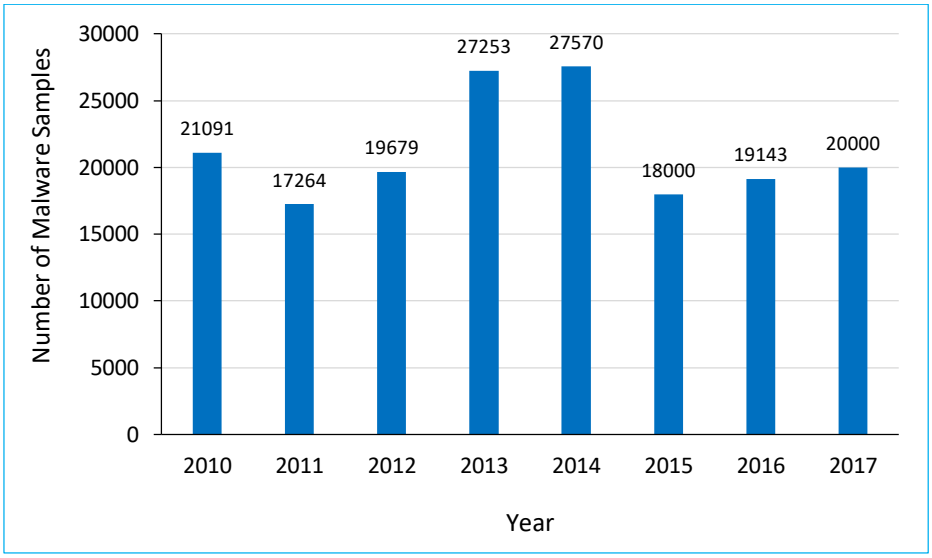


Figure 4.2: Year-wise distribution of malware samples in the dataset.

4.2.1.1 Data Cleaning and Labelling

The key steps in data preprocessing includes data cleaning, labelling, transforming raw data into numerical values, and data normalization. The data cleaning step involves removing the samples having all null values and filtering the unique malware binaries based on MD5 hashes. The next step is data labelling which helps to train and evaluate a model at low FPRs. In the dataset, VirusTotal provides the scan result of 57 (66 at present) antivirus detectors. It is pertinent to understand that the number of detections do not remain static over time, though number of positives usually increase after rescanning of binaries. We have adopted a voting method to decide to label a malicious sample and set a threshold of five detections to label a malicious sample because it is rare to find false positives from five or more detectors. Using this threshold, only six samples were discarded in the dataset.

4.2.2 Big Data Framework

In order to analyze the behavioral trends of malware data at large scale, a scalable architecture has been built on the top of Apache Spark. The main components of proposed architecture include Apache Spark, HDFS for distributed storage, Hadoop YARN, and Python for data-intensive analysis. All software components are open source from Apache Software Foundation (www.apache.org), a non-profit public charity organization. While Apache Spark has already been discussed in Section 1.4.2, the other free open source software (FOSS) components are discussed below:

- **HDFS:** HDFS, one of the main components of Apache Hadoop ecosystem, is used for distributed storage. HDFS runs on commodity hardware and shares many resemblances to existing distributed file systems. It supports large data files and can scale a cluster to hundreds of nodes. The top benefits are highly fault-tolerance, low-cost hardware, and portability across heterogeneous hardware and software platforms.

- **Apache YARN:** It is an open-source cluster management system which is easy to deploy and provides better cluster utilization. It is suitable for an environment with high security as it protects the cluster from malicious code and clients.

- **Development Language:** Python is a general purpose language, which is very easy to understand and code. It is highly expressive language, which means it needs far fewer lines of code as compared to languages like C++ or Java. It is used for data-intensive analysis and is preferred as development language due to two compelling reasons: 1) availability of rich ecosystem of scientific libraries, and 2) almost all important deep learning and machine learning research articles use Python for implementation. Jupyter notebook provides the development environment. PySpark is the Python API for Apache Spark.

4.2.3 Functional Flow

In order to have better understanding of the proposed architecture, the following steps (as shown in Figure 4.3) elaborate the functional flow of malware behavioral trends analysis:

1) Malware data is collected from widely used diverse repositories.
2) Malware samples are decompressed, filtered, cleaned and integrated to obtain unique malicious binaries.

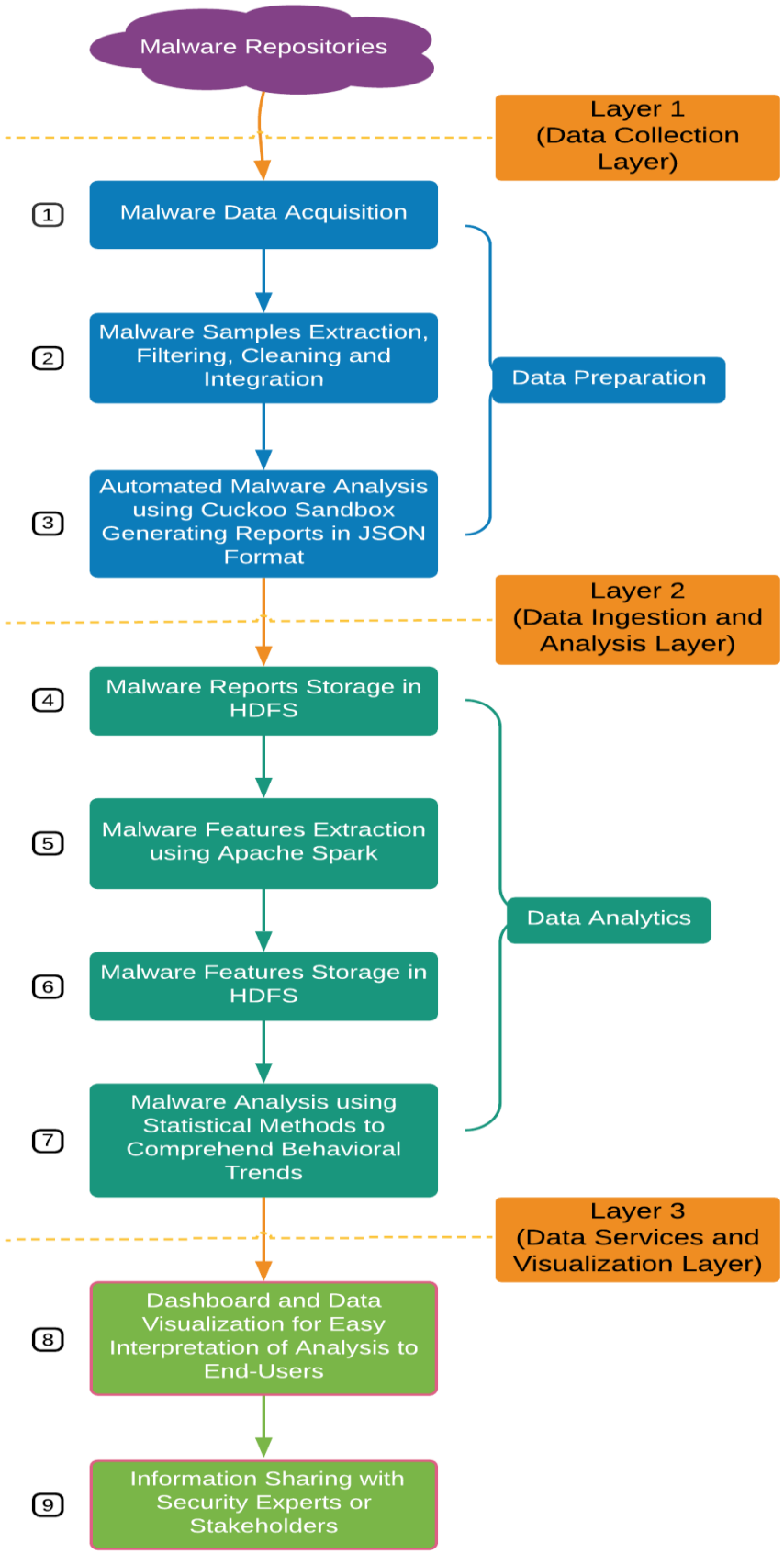


Figure 4.3: Functional flow of malware trends analysis.

3) Cuckoo Sandbox is used to perform automated malware analysis, which generates malware analysis reports in JSON format.
4) The analysis reports in JSON format are stored in HDFS (on a local cluster).
5) On the top of Apache Spark, the JSON reports are processed and analyzed to extract the malware features, which are used to understand their behavior and future trends.
6) The extracted features are stored back to HDFS.
7) Malware data analysis is performed using statistical methods to have a comprehensive understanding of malware behavior along with the changing trends over a period 2010 to 2017. The analysis can lead to useful predictions about the future malware threats.
8) The dashboard helps to view the details of the statistical analysis of malware attributes (like malicious binary size, packers used, network activities, and registry operations). Data visualisation transforms the analytic results into diagrams and tables, which provides easy interpretation of the analysis for end-users or decision makers.
9) The information about behavioral trends can be shared with big data security experts and other stakeholders to draw threat intelligence, which enables the security analysts to come out with preventive measures to handle future threats.

4.3 Malware Trends Analysis

The subsequent sub-sections discuss the results obtained though statistical analysis of malicious binaries for the following features which are extracted using static and dynamic analysis:

- Malicious binary metadata: It includes the corresponding MD5, file name, file size, and timestamp of malware creation.
- Commonly used packers: This includes the name of the packer which is used to compress and/or encrypt its content.
- Network activities: This provides the information about the protocols used by malicious programs for establishing a connection with their command and control (C&C) server, IP addresses and domains.
- Registry activities: These include the registries accessed, read/created, modified, and deleted along with their key-values.

4.3.1 Experimental Setup

The experiments were conducted on Apache Spark cluster comprising of five machines – one acting as a master and remaining four as slaves. All machines are having Ubuntu 16.04.2 as operating system. The configuration of the master machine is Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD. All slave machines are having the same configuration: Intel(R) Core TM i3-4160 CPU @3.60 GHz, 8 GB DDR3 1600 MHz, and 1 TB HDD. Table 4.1 provides the details of experimental environment.

Table 4.1: Experimental environment.

<table>
<tr><th>Category</th><th colspan="2">Description</th></tr>
<tr><td rowspan="5">Software Tools</td><td>Apache Spark</td><td>Version: 2.1.0</td></tr>
<tr><td>Apache YARN</td><td rowspan="2">Version: 2.7.3</td></tr>
<tr><td>Apache HDFS</td></tr>
<tr><td>Python</td><td>Version: 3.6.0</td></tr>
<tr><td>Ubuntu</td><td>Version: 16.04.2</td></tr>
<tr><td rowspan="2">Hardware Configuration</td><td colspan="2">Master: Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD</td></tr>
<tr><td colspan="2">Slaves (04): Intel(R) Core TM i3-4160 CPU @3.60 GHz, 8 GB DDR3 1600 MHz, and 1 TB HDD</td></tr>
<tr><td>Dataset</td><td colspan="2">Malware samples: 1,70,000</td></tr>
</table>

4.3.2 Malware Binaries Size

The present-day malware can propagate more quickly (as these make use of the Internet), unlike earlier malware binaries which were typically spread through physical media like floppy disks. In fact, malware authors tend to design malicious binaries in such a way so that these propagate within a fraction of a second. In the dataset, most of the malicious files are found to be of small size because it helps these to transmit easily and get executed quickly.

A statistical analysis of the size of malicious files in the dataset is performed. The mean size ($\bar{x}$) is computed using equation (4.1).

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n} = \frac{x_1 + x_2 \ldots\ldots x_n}{n} \qquad \cdots (4.1)$$

where x_i is the size of i[th] malicious file and n is a total number of such files.

Variability in the size of malicious files is measured by using standard deviation (SD), which is computed using equation (4.2).

$$SD = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}} \qquad \cdots (4.2)$$

The mean size and standard deviation of the malicious samples considered in the dataset are 362.2 kilobytes and 844.2 kilobytes respectively. The value of standard deviation is much more than the value of the mean which reveals that there is randomness in the data and the size of malicious files is more spread out. On further analysis, it is perceived (Figure 4.4) that more than 77% of the malicious files are of the size smaller than 400 kilobytes and only 5.84% of files have the size greater than 1 megabytes.

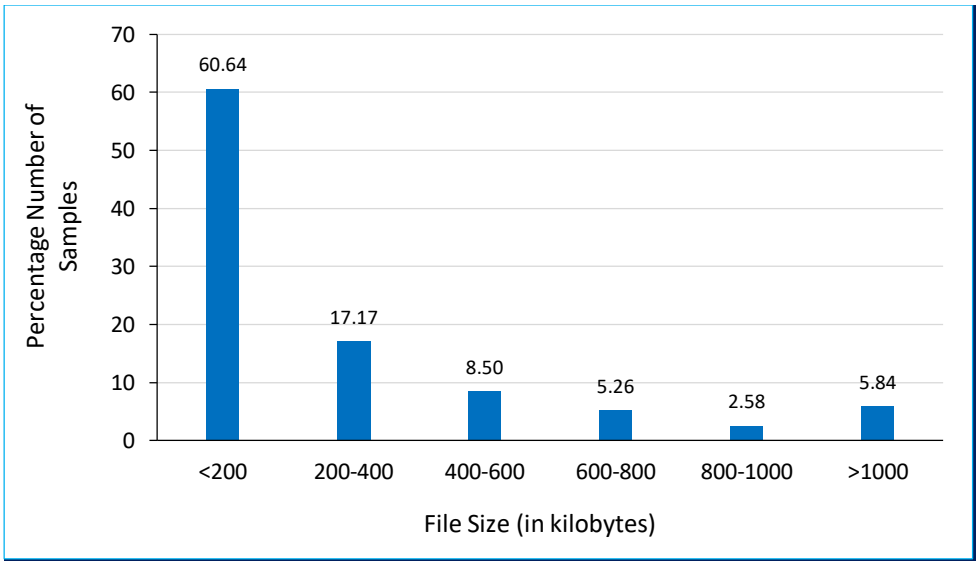


Figure 4.4: Malware binaries size vs. percentage number of samples.

In order to have a closer analysis of file size spread, the five-number summary is computed, i.e., minimum, Q_1 (1st quartile or 25th percentile), Q_2 (2nd quartile or median), Q_3 (3rd quartile or 75th percentile) and maximum size of a malicious sample. The malicious binary size when arranged in the numerical order, the intermediate term divides the whole set of observation into two halves. The value of this intermediate term is called the median or the 2nd quartile. The value of intermediate term between the first term and the median, is called 1st quartile. The value of the intermediate term between the median and the last term is called 3rd quartile. Equations (4.3), (4.4) and (4.5) show the way to compute these values.

$$Q_2 = Median = \begin{cases} (x_{\frac{(n+1)}{2}}), & if\ n\ is\ odd \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right), & if\ n\ is\ even \end{cases} \quad \cdots (4.3)$$

$$Q_1 = \left(x_{\frac{n+1}{4}}\right) \quad \cdots (4.4)$$

$$Q_3 = {}^{x_{\frac{3(n+1)}{4}}} \quad \cdots (4.5)$$

Table 4.2 shows five-number summary for the size of malicious files grouped by year of their discovery. It presents the fact that for different years, 75% of the samples are less than or equal to the size specified under the column heading Q_3 (75th percentile).

Table 4.2: Five-number summary (in kilobytes) of malware samples.

Year	Minimum	Q_1 (25th Percentile)	Q_2 (Median)	Q_3 (75th Percentile)	Maximum
2010	0.708	73.768	159.14	434.176	12534.46
2011	0.512	45.056	118.902	319.488	29388.89
2012	0.571	67.584	159.744	365.851	32442.37
2013	0.761	84.72675	180.328	354.2628	14484.79
2014	1.024	149.382	323.648	509.35	30638.08
2015	0.651	86.486	256.273	549.648	13164.61
2016	0.842	92.121	302.602	407.763	23267.91
2017	0.732	98.974	312.948	510.291	32711.33

Figure 4.5 presents the mean size of malware grouped by their year of discovery. It illustrates that the average size of malicious files is increasing from 2010 to 2017. This escalation in size is obvious due to the reason that malware authors tend to incorporate more and more functionalities, thus making these more complex and bulky. Malware size may also be affected due to the use of different encrypting and packing techniques being used by malware authors. Moreover, the output of compiler used to build malicious binaries is also getting bulky due to feature rich inbuilt libraries. The other contributing factors are ever increasing computer memory, disk space and Internet bandwidth.

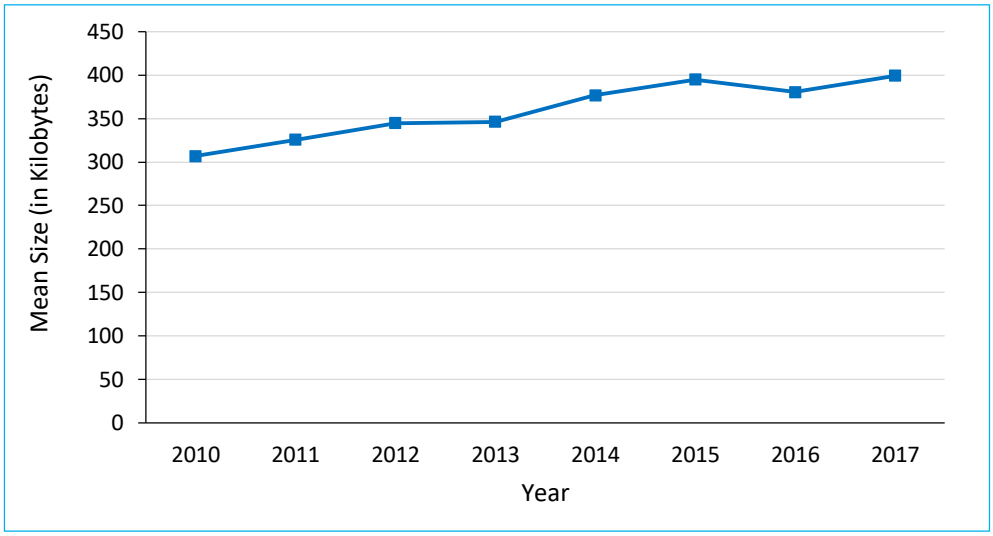


Figure 4.5: Trend line of mean size of malicious samples.

4.3.3 Commonly used Malware Packers

A packer is a covering laid around a piece of software to compress and/or encrypt its content. These can be used by any software to minimize download time or storage space and are extremely popular with malware authors because these help them to hide the malicious files from malware scanners, complicating malware analysis, and shrinking the size of malicious executables. Antivirus products must have the capability to unpack the packed programs in order to examine the concealed payloads using various reverse engineering tools.

In the dataset, out of total analyzed files, 30.61 % are found to be packed according to PEid (a signature-based packer/compiler/cryptor detector) [164]. Figure 4.6 presents the percentage number of malicious samples packed using different packers for the complete dataset. It reveals that UPX (Ultimate Packer for Executable) is the most commonly used

packer and 44.87% of packed files use it. The obvious reason for such a high usage may be attributed to the fact that it is an open source packer and has a very good compression ratio. The second most popular packer used by malware authors is Armadillo (which packed 11.91% of the files) followed by Nullsoft Installer.

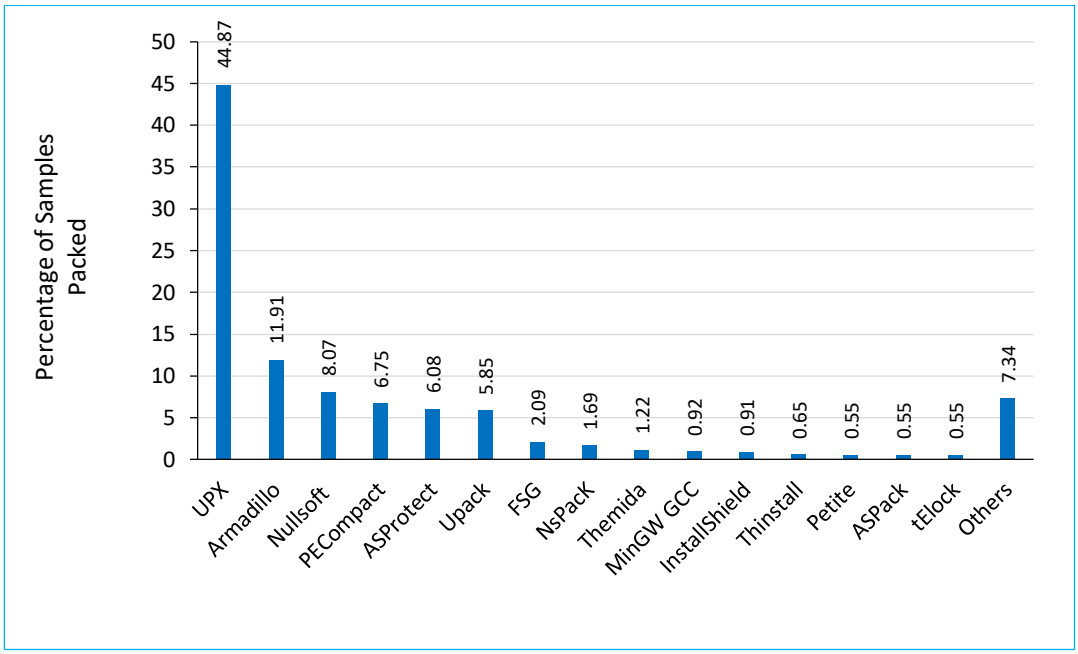


Figure 4.6: Packers used vs. percentage number of malware binaries.

From the analysis of the year-wise trend of the packed malware (Figure 4.7), it is apparent that the percentage number of samples, which are packed by any of the packers, is consistently decreasing with time. There may be two reasons for it. First, the malware authors are making use of customized packers which the existing tools are not able to detect (as these are signature-based) and second, the malware authors are coming up with other innovative techniques to evade detection of malware rather than using packers for this purpose.

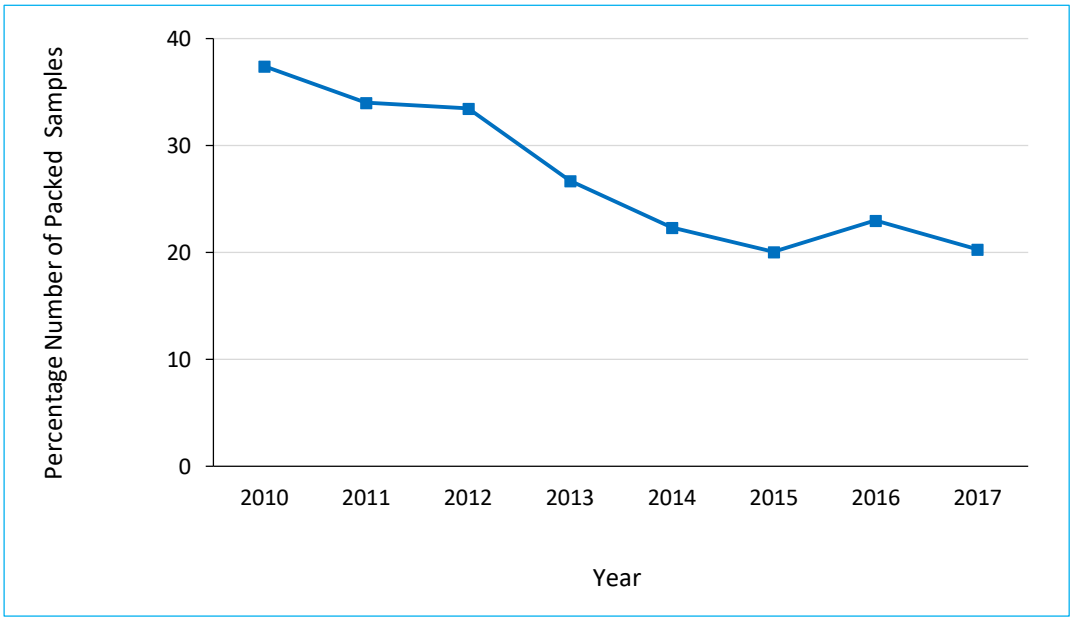


Figure 4.7: Trend line of packed malware binaries.

Figure 4.8 presents the year-wise trend of packers used by malware authors. It indicates that for every year, UPX is used as a top packer among all and its usage is gradually increasing. Use of Nullsoft also shows a notable rise. The other packers like Armadillo, Upack, etc. are becoming less popular among malware authors.

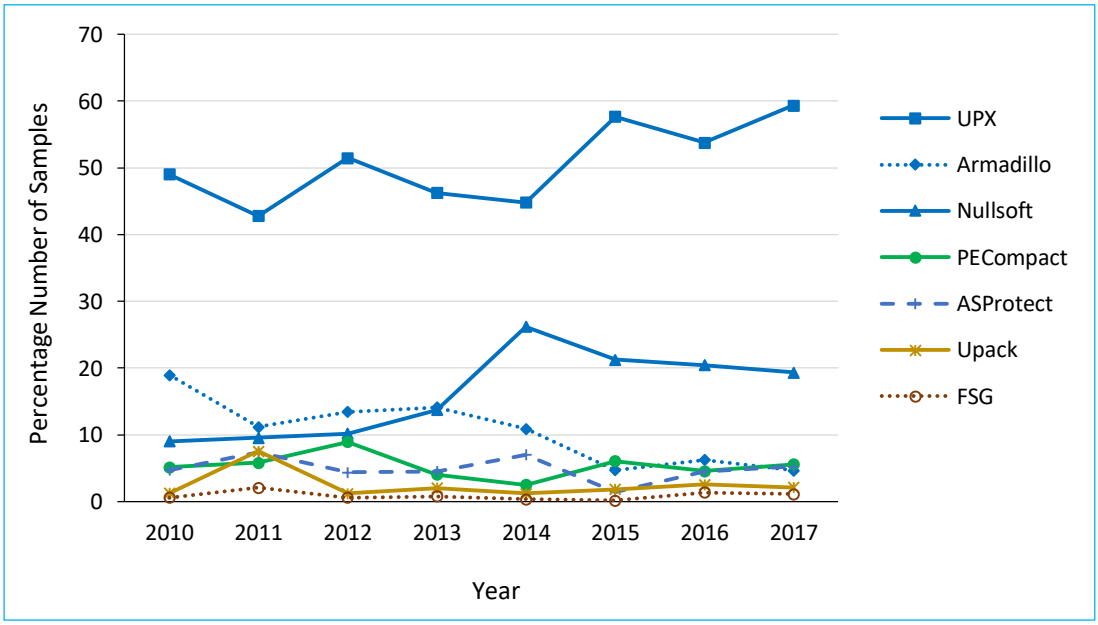


Figure 4.8: Trend line of various packers used by malicious samples.

By taking advantage of these packers, attackers can thwart signature-based scanners. Even if an antivirus scanner has an updated database containing the signatures of the malicious files, it will not be able to detect the newly packed version of the known malicious sample. A major challenge for big data security researchers is to overcome the composite anti-decompiler and anti-debugging strategies of these packers which in turn pose significant challenges for existing reverse engineering and dynamic analysis tools while spotting potential malicious threats.

4.3.4 Network Activities

The reports generated by Cuckoo Sandbox provide the information about the network behavior of malicious programs for establishing a connection with their C&C server, IP addresses and domains. Figure 4.9 provides the visual representation of the network behavioral activities observed during the analysis. It specifies that most of the malware samples make use of TCP for carrying out communication over the Internet. A significant number of malware samples use UDP. The reason may be attributed to the fact that the traffic based on UDP is usually overlooked by organizations and thus, malware authors exploit it to make entry of malicious programs into the victim organizations.

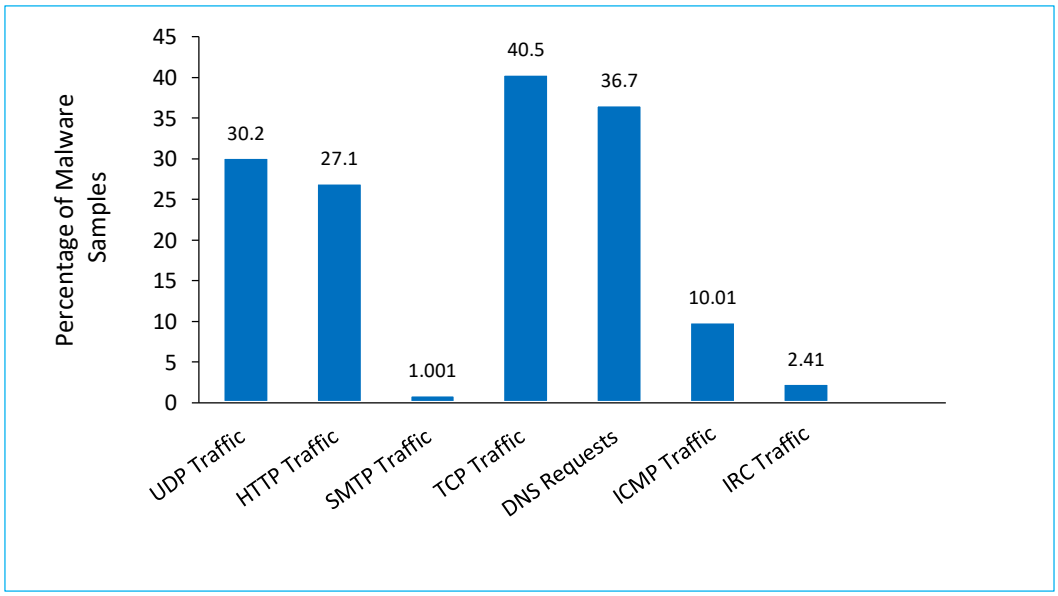


Figure 4.9: Percentage of malware samples showing network behavior.

Further, around 37% of the samples associated with some network activity make a query to the DNS server for resolving a domain name. The ICMP activity of malware reveals

that before selecting a candidate for establishing a connection, a malware sample checks if the remote host is accessible or not. Figure 4.10 depicts the usage of HTTP, SMTP, IRC protocols among malware authors over the years. It shows that the number of malware samples using HTTP are increasing, but the use of IRC is decreasing with time. It is a challenge for researchers to analyze the network traffic which is growing in velocity, volume and variety. Also there is a need to develop automated big data tools for analyzing network traffic, which can quickly identify suspicious activities.

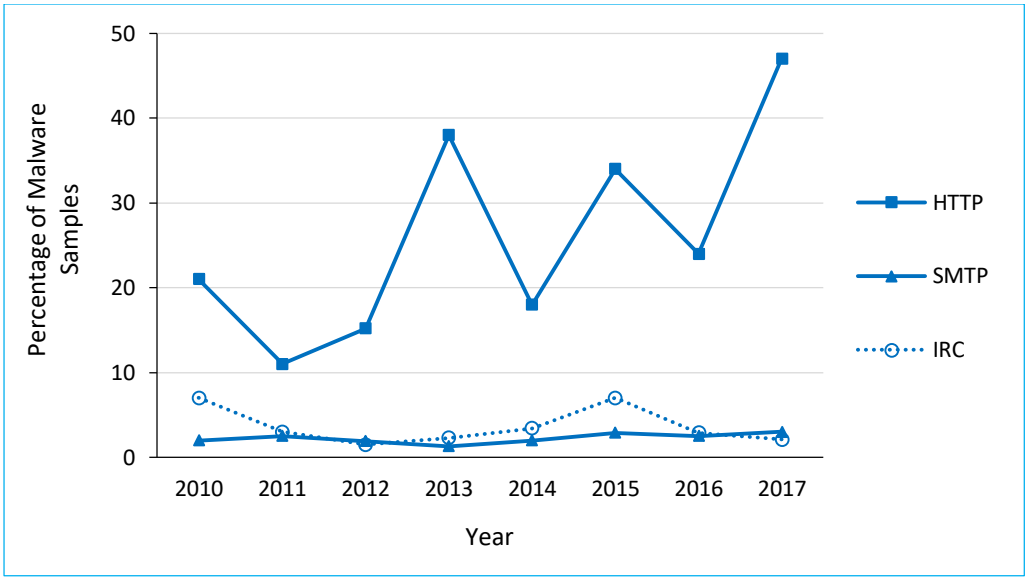


Figure 4.10: Usage of HTTP, SMTP, and IRC in malware samples.

4.3.5 Windows Registry Activities

The Windows operating system is composed of a huge hierarchical database to store information about system configuration known as registry. Registry analysis aims to extract relevant information as well as the context which modifies or creates data and can also help to understand the malware behavior. The registry activities performed by a malware can disclose a lot of information such as signs of infection, modifications done to the system or its applications and evidences of persistence mechanisms used. Registry entries are divided into a number of different logical sections mostly dependent on the type of functionalities they perform.

85.72% of the malware samples access one or more registry entries. 82.6% of the samples read/create registry entries. A few malware samples (0.09%) generate an entry under ‘HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates\TrustedPubli

sher\Certificates', which is used for installing its own certificate as reliable. Around 15.37% of the malware samples are deleting one or more registry. Usually, malware deletes Windows services related to security so that the security software will no longer function properly. In the dataset, 0.65% of malicious samples attempt to delete the registry under the key 'HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\ Control\SafeBoot' to prevent the user from booting into the safe mode. A significant number of malware samples (53.43%) modify the registry keys. For example, a malicious program can disable notifications/warnings about the security center by modifying the registry key values under 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ Security Center'. Once such notifications get disabled, the system cannot detect or notify the malware.

Figure 4.11 depicts the year-wise analysis of registry activities performed by malicious programs. It shows that a substantial number of malware access the registries. Most of them read/create the existing/new registry entries and some of them even modify these for successful execution of malicious programs. It further indicates that a small number of malware samples perform the deletion activity on registry entries.

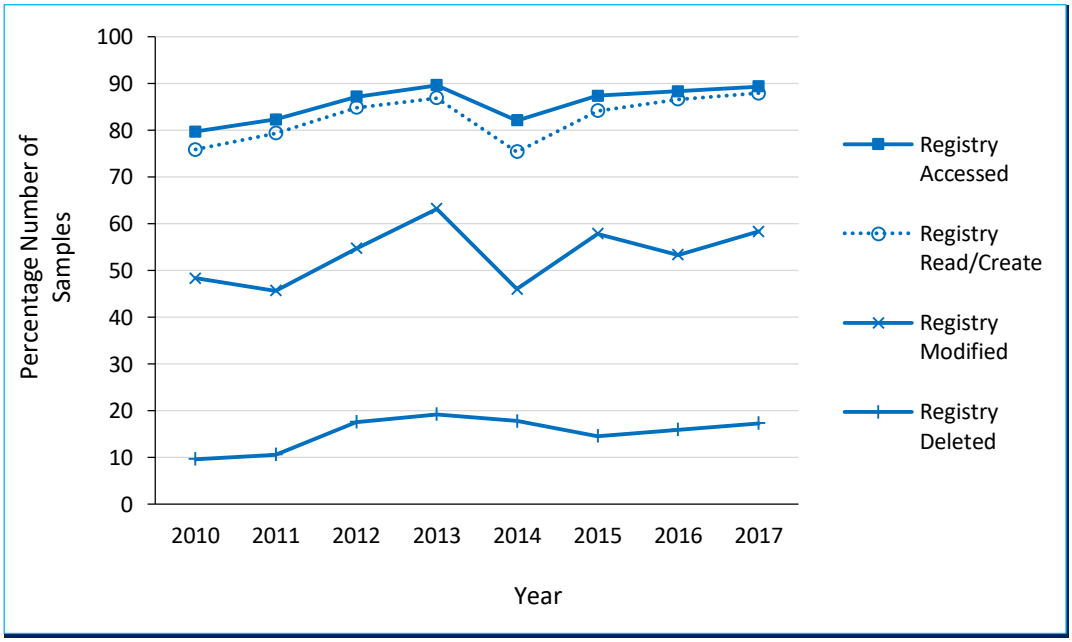


Figure 4.11: Comparison of registry activities.

Another set of malicious programs modifies the registry for realizing persistence. Such programs make use of various Windows registries in different ways to auto-start

themselves at boot time. About 30% of malware modify registry keys to get them launch during startup. It shows that the most common registry used to achieve persistence is 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run'. Around 7.63% of malware samples use it for getting started during boot time.

Malware can also be hooked to a particular Winlogon event under the key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ Winlogon'. About 0.32% of malware samples use this auto-start location to achieve persistence. Under Winlogon, a value Userinit is modified by malware (1.57%) to get auto-start at boot time. About 0.58% of the malware samples modify the key under 'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services', which contains the information related to the Windows services configuration. Another popular location which is used by malware authors to achieve persistence is AppInit_DLLs. The DLL values are stored at 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT \CurrentVersion\Windows' and are loaded using User32.dll, which in turn is used by almost every process. Thus, malware authors use this location most of the time. About 0.99% of malware samples modify this registry in the dataset. Usually, malware authors install their malicious code as a service on Windows operating system by making use of svchost.exe. A list of processes is contained under 'HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost', which runs as services under svchost.exe program.

Though the binary structure of the registry system remains same across the successive versions of Windows operating system, some of the keys and values change significantly. The security analysts have to identify specific registry keys and values in context to the malicious programs based on a particular version of operating system. It is yet another challenge for the big data security researchers to understand the changes, and quickly update the analysis techniques and procedures according to these changes in Windows registries.

4.4 Discussion and Key Findings

The malware data is growing continuously at an astonishing rate and has now been recognized as a big data problem. It is in the interest of any security organization to have a scalable system to analyze malware binaries at large scale. Keeping in mind this concern, the proposed framework supports efficient processing of malware binaries on

the top of Apache Spark and HDFS in a distributed environment. It is imperative to understand that malware detection and removal alone is not sufficient at all. It is equally important to extensively comprehend the changing behavior of malicious binaries in order to extract threat intelligence.

Malware binaries size keeps on changing as malware authors frequently change packing or encryption techniques. Moreover, the size can also vary as these may include junk data to avoid detection. Some antivirus vendors have fixed the upper limit on the size of malware file aiming to improve their scanning performance. The changing trends about the size of malicious binaries can help such vendors to update their upper size limits at any time. Such trends allow security experts to comprehend the threat landscape to provide actionable cyber security intelligence in order to stay ahead of any potential threat. A packer allows malware code to be less detectable as a smaller code is hard to detect. It seems that the anti-packer tools are unable to identify the type of packers and thus, it is a key challenge for security researchers to develop efficient packer detection tools. The network activities provide the information about the protocols used by malicious programs. There is a need to develop automated big data security solutions which can identify the suspicious activities in huge network traffic to strengthen threat intelligence. The registry analysis provides a lot of information including the changes made by malicious binary and evidence of persistence mechanisms.

A thorough understanding of the behavioral trends of historical and current malware binaries allow security experts to focus on prevalent areas immediately. The threat intelligence so generated can be shared with various stakeholders to issue early warnings and preventive measures, which can be considered to handle future threats. Besides, the security experts can devise their strategies and issue advisories for setting up budgets for essential preventive measures. Moreover, monitoring such behavior trends can help big data security experts to proactively take performance-related decisions and also facilitate them in selecting behavioral features for designing and developing advanced malware detection systems.

4.5 Summary

This chapter discussed the proposed scalable architecture for trends analysis of malware samples generated by Cuckoo Sandbox on the top of Apache Spark and HDFS. Analysis of some of the prominent malware attributes (like malicious binary size, packers used,

network activities, and registry operations) is performed using the proposed architecture. The analysis has shown the trends related to malware behavior which can help the researchers and experts to explore the potential areas for understanding the threat landscape and combating malware threats. These trends can be further used by big data security experts to draw cyber threat intelligence, which may be shared with stakeholders to implement proactive security strategies in order to deal with future threats.

Chapter 5: Big Data Framework for Zero-Day Malware Classification

Malware has already been recognized as one of the most dominant cyber threats on the Internet today. It is growing exponentially in terms of volume, variety and velocity and thus overwhelms the traditional approaches employed for malware detection and classification. Moreover, with the advent of IoT, there is a huge growth in the volume of digital devices and in such scenario, malicious binaries are bound to grow even faster making it a big data problem. In order to analyze and detect unknown malware on a large scale, security analysts need to make use of machine learning algorithms along with big data technologies. These technologies help them to deal with current threat landscape consisting of complex and large flux of malicious binaries.

This chapter presents a proposed framework built on the top of Apache Spark for zero-day malware classification using machine learning. An integrated set of static and dynamic features is created by extracting the relevant information from the JSON reports. Different machine learning models are built for classification of malware and compared to choose the best model for our dataset on the basis of evaluation metrics like FPR, FNR and accuracy. This chapter partly accomplishes objectives 2 and 3.

5.1 Introduction

Though the area of machine learning has been developed towards timely detection of unknown malware, but it faces challenges due to advancement of malware data as the attackers are constantly coming up with a large number of malware samples along with new techniques to fool the detectors.

With this threat landscape, the malware detection has become a big data problem. In recent times, big data analytics has got considerable attention of security researchers and

practitioners. Big data analytics, machine learning, and other decision-making techniques, along with augment human interface, aim to reduce the response time and increase the effectiveness in detecting zero-day malware threats. It can help in updating the anti-malware solutions near real time to deal with new malware threats. The historical data has its own significance and can provide cyber intelligence to deal with future threats.

Earlier, Apache Hadoop has become the de facto standard for big data implementations. Presently, Apache Spark has got momentum due to ease-of-use and better performance than Apache Hadoop, and is one of the most active Apache projects in big data. The proposed framework is implemented on the top of Apache Spark and uses its scalable machine learning library (MLlib) for malware detection at large scale.

5.2 Preliminaries

We used three base classifiers, namely, NB, SVM, and RF. Each of these classifiers belongs to different family of classifiers and thus uses different approaches to classify the input data. The description of each of these base classifiers is given below:

5.2.1 Naïve Bayes

NB classifier is based on Bayes' theorem with strong independent assumptions. It predicts the probabilities of a given instance belonging to a class in a dataset. It assumes that the occurrence of a feature in a class is independent of any other feature i.e., all features are assumed to contribute independently in calculating the probability for classification of data [165]. This model is useful for very large datasets and is easy to build. Let the dataset D has a feature vector$X = (x_1, x_2, \dots\dots\dots x_n)$, then according to Bayes' theorem, the posterior probability is computed by using the equation (5.1).

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \qquad \dots (5.1)$$

where $P(c|x)$ is the posterior probability of class c, given predictor attribute x. $P(x|c)$ is the probability of predictor attribute of class c. $P(c)$ is the prior probability of class c and $P(x)$ is the prior probability of predictor attribute.

5.2.2 Support Vector Machine

SVM is a supervised machine learning algorithm in which each data point is plotted in n-dimensional space, where n represents number of features. Each feature values represents the value of a particular coordinate. Then, the classification is performed by identifying a hyper-plane which differentiates the data of one class from another one. Training instances known as support vectors and margins defined by them are used to recognize the hyper-plane [166].

5.2.3 Random Forest

It is an ensemble method consisting of several decision trees and a technique called bagging (bootstrap aggregation). Bagging involves training each decision tree on a subset of the original dataset which is created by sampling with replacement. The final class in RF is calculated using the majority voting on the result of all decision trees. It is highly efficient algorithm that is effective on a large dataset [167].

5.3 Proposed Architecture for Zero-Day Malware Detection

In this section, a scalable architecture for zero-day malware detection is proposed. This architecture is designed on the top of Apache Spark for distributed data processing along with its machine learning library (MLlib) for scalable malware classification. Figure 5.1 depicts the block diagram of proposed scalable architecture and its various components are discussed in following sub-sections.

The proposed platform helps in:

- Extracting the features from JSON files
- Studying the impact of extracted features on malware classification
- Malware detection using scalable machine learning library (MLlib)

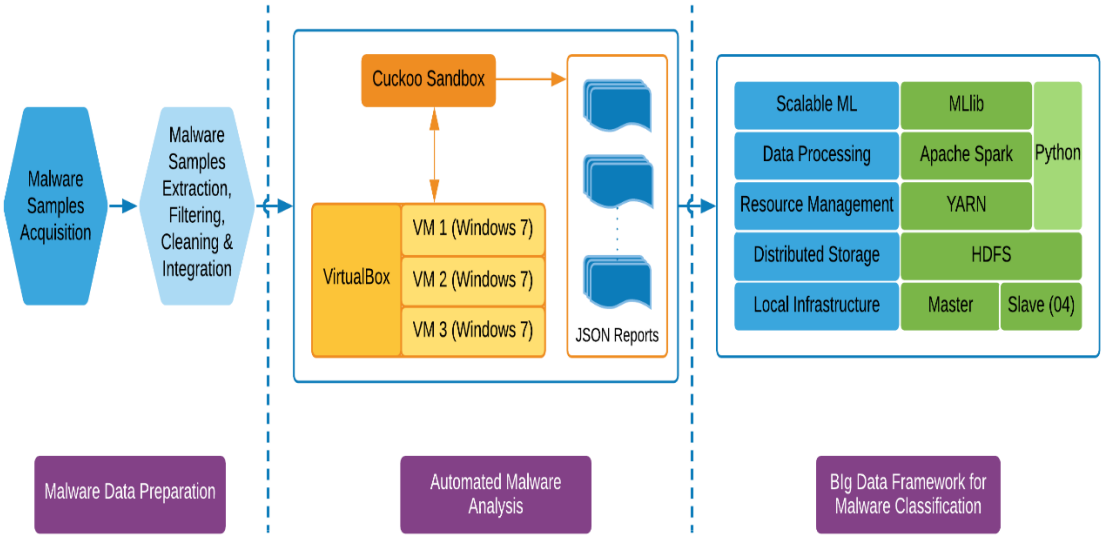


Figure 5.1: Block diagram of proposed architecture for zero-day malware detection.

5.3.1 Feature Extraction and Aggregation

Machine learning and deep learning algorithms generally work on numeric feature space, where rows and columns represent instances and features respectively, rather than raw attributes data. Feature extraction or representation learning denotes transformation of the raw data into a feature vector. The process of transforming raw data into feature space is commonly known as vectorization. The vectorization of variables speeds up the performance of learning algorithms because it performs highly optimized numerical operations (like matrix-matrix addition or multiplication, matrix-vector multiplication) together for all samples in a dataset. Therefore, the vectorized code extracts more parallelism resulting into less computational overhead. The commonly vectorization techniques are categorical, free-form string, ordinal, and sequential vectorization [132].

In general, the diverse feature set can help to understand the characteristics of a malicious application and thus, it is better to consider static and dynamic features to perform malware detection more precisely. We have identified an integrated set of static and dynamic features based on state of art malware classification and our dataset. Thereafter, these features are converted to an aggregated feature vector which is fed into neural network. In all, the combined feature vector has dimensionality of 1269. Since the count or frequency values can vary considerably and thus, min-max scaling is used to scale the feature values into a fixed range of [0, 1] using equation (5.2):

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad \ldots (5.2)$$

where x, x', x_{min} ,and x_{max} represent original, normalized, minimum and maximum values of a feature. This kind of scaling of values is a general requirement for comparing the performance of machine learning and deep learning algorithms. Most of the machine learning classifiers consider Euclidean distance to compute the distance between two points and thus, it is necessary to normalize the feature values so that each feature contribute proportionately to it. In deep learning, if feature values are on different scales then the gradient descent may update faster for some features than others as weight updates are dependent on the feature values [168]. The various features are described below and summarized later in Table 5.1:

5.3.1.1 File Metadata

The metadata of malware binary includes the information like file size, FileVersion, ProductVersion, ProductName, CompanyName etc. Unlike benign file, malware authors avoid putting metadata. Since the information about the metadata consists of large strings, so the metadata feature is considered as Boolean having value 1 if metadata is available, and 0 otherwise.

5.3.1.2 File Size

Malware authors tend to keep the size of a malicious binary small. In the dataset, around 75% of binaries are having size less than 400 KB and only 5% have size more than 1 MB. To represent file size, the feature vector consists of integral values denoting the size range in kilobytes: 0-400, 400-800, and 800 or more.

5.3.1.3 Packer Detection

Packers are extremely popular among malware authors to compress and/or encrypt its content to minimize the download time and storage space. It changes the byte structure of the malware and thus, can bypass the signature based detection. In the dataset, the top three packers used by malware authors are UPX (Ultimate Packer for Executable), Armadillo and Nullsoft Installer. Cuckoo Sandbox internally uses PEiD signature database with YARA (http://virustotal.github.io/yara/) to collect packers information. In

the feature set, packer takes Boolean value, i.e., 0 if there is no packer, and 1 if packer is used.

5.3.1.4 Sections' Information

A section name is a good indicator of malware. A PE file has standard sections like .text, .data, .reloc, .rsrc, etc. Each section has a virtual size (i.e. memory assigned during loading process) and raw data size (size of the section on disk). The JSON reports generated by Cuckoo Sandbox contains both of these sizes. If there is large difference between these sizes, the related section is considered as suspicious which may be due to packing or encryption of the section.

Entropy represents a measure of efficiency of information storage [169]. This implies that a packed file possesses high entropy. The entropy is calculated as quantity of bits per byte and thus, the maximum value of entropy is 8.0. The JSON reports generated by Cuckoo Sandbox contains the entropy value for different sections of the PE file. The sections for which the value of entropy is greater than 7.0 are considered as suspicious sections. In the feature set, we considered number of suspicious sections as a feature.

5.3.1.5 Dynamically Linked Libraries (DLLs)

Most of Windows APIs are implemented using DLLs. When a process calls an API, it must load the related DLL into its own memory space. This fact is exploited by malware authors by distributing malware as DLL which allows malware to run inside any process and also gives access to complete addressable host or target memory. Thus, DLL is a predominant feature for malware detection as it reveals the resource information for a program. The most frequently used DLLs include kernel32.dll, shell32.dll, user32.dll, urlmon.dll, ntdll.dll, gdi32.dll, ole32.dll, version.dll, comctl32.dll, and advapi32.dll. To create DLLs feature vector, we have prepared a master list of 728 unique DLLs to create a frequency vector for DLLs.

5.3.1.6 Dropped Files

Malware drops some files on the system during its execution to avoid its detection. Malware authors keep on employing innovative solutions to deceive antivirus applications, e.g., a benign file may be dropped to install malware on host machine. In the feature set, we have considered number of dropped files.

5.3.1.7 Windows API Calls

Windows API call is one of the most dominant feature for knowing malware behaviour. It shows the manner in which malicious binaries interact with Windows libraries [170]. The JSON reports generated by Cuckoo Sandbox include the name of functions invoked, call sequence, argument details, and their return values. It gives large number of API calls and thus, we have used a master list of 794 unique APIs [171] which are useful to distinguish malware and benign files, and created a frequency vector whose elements denote the number of times (i.e. frequency or count) a particular API is called. The API features are further reduced to 512 after removing the APIs having zero count.

5.3.1.8 Mutex Operations

Mutexes ensure that only a single copy of malware is running in the system. Mutexes are helpful in malware analysis as these use hard coded names to ensure uniqueness of malware making them good host based indicators. In general, malware use CreateMutex function to create a mutex and try to open an existing mutex using OpenMutex function with same name to ensure that only one version of malware is running at a time.

5.3.1.9 File Activities

File activity helps in understanding the role of a malware. For example, if the malware creates a file to store browsing history, it is a kind of spyware. The file activities can provide information about number of files accessed, read, created, modified, and deleted. We have built the frequency vector for all these operations.

5.3.1.10 Registry Activities

The Windows registry consists of a large hierarchical database to store information about system configuration. Registry is one of the most common targets used by malware to introduce susceptibilities by reading, creating, modifying, or deleting the registry keys. The registry activities helps to know about signs of infection, modifications done to the system and evidences of persistence mechanisms used. In the dataset, more than 80% of the malware samples access, read or create one or more registry entries. Around 15% of malware are deleting the one or more registry keys and a large number of malware (around 50%) modify the registry keys.

Table 5.1: Selected features and their values.

Types	Category	Feature Count	Features	Raw Type	Derived Type
Static Features	File Metadata	1	File metadata present	String	Boolean
	File Size	1	File size between 0 to 400 kb	Integer	Integer
		1	File size between 400 to 800 kb		
		1	File size greater than 800 kb		
	Packer Detection	1	Packer used	String	Boolean
	Sections' Information	1	No. of suspicious sections	String	Integer
Dynamic Features	DLLs	512	Frequency of DLLs	String	Integer
	Dropped Files	1	No. of dropped files	Integer	Integer
	Windows API Calls	728	Frequency of API calls	String	Integer
	Mutex Operations	1	No. of mutexes	String	Integer
	File Activities	1	No. of files accessed	String	Integer
		1	No. of files read or created		
		1	No. of files modified		
		1	No. of files deleted		
	Registry Activities	1	No. of registries accessed	String	Integer
		1	No. of registries read/created		
		1	No. of registries modified		
		1	No. of registries deleted		
	Network Activities	1	No. of TCP connections	String	Integer
		1	No. of ICMP connections		
		1	No. of UDP connections		
		1	No. of IRC connections		
		1	No. of HTTP connections		
		1	No. of SMTP connections		
		1	No. of DNS requests		
		1	No. of hosts requests		
		1	No. of domains		
		1	No. of IP addresses		
	Process Activities	1	No. of processes created	String	Integer
		1	No. of processes modified		
		1	No. of processes terminated		

5.3.1.11 Network Activities

The Cuckoo Sandbox provides the summarization of all incoming and outgoing network traffic such as TCP, UDP, IRC, SMTP, HTTP, ICMP traffic, and DNS requests. The network connections can help to identify the ports on which malware is listening. In the dataset, the majority of malicious binaries use TCP for communicating through Internet. The feature vector comprises of the features to denote the count for each network connection.

5.3.1.12 Process Activities

Process activities can show whether the malware has created, modified or terminated one or more processes. Creating or modifying an existing process can help malware to run outside the current program.

5.3.2 Big Data Tools

A big data framework is developed on the top of Apache Spark for classification of malicious binaries. The other components of the proposed framework includes Spark's scalable MLlib, Apache YARN as resource manager, HDFS for distributed storage, and Python language for data analysis implementation. Apache Spark and its scalable machine learning library are discussed in Sections 1.4.2 and 2.3.1 respectively. The other software components are discussed in Section 4.2.2.

5.3.3 Functional Flow

Following steps elaborate the functional flow (shown in Figure 5.2) of proposed big data framework:

- Step 1: The analysis reports generated during automated malware analysis (in JSON format) are stored in HDFS (on a local cluster).
- Step 2: The JSON reports are processed and analyzed using Python on the top of Apache Spark to extract static as well as dynamic features of malware. These are stored into HDFS.
- Step 3: The impact of extracted static and dynamic features is studied to underline their importance in malware classification.

- Step 4: A classification model is built to accomplish zero-day malware detection by training with labelled dataset using 10-fold cross validation. In 10-fold cross validation, the original dataset is randomly partitioned into 10 data subsets (DS) of equal size. Out of these, one is used for data validation and remaining 9 subsets are used for training purpose. This process is repeated 10 times so that each subset is used exactly once as the validation data. The results are averaged to obtain a single estimation. Three machine learning algorithms are used, namely, NB, SVM and RF. Thereafter, the classification results are analyzed visually and evaluated using various evaluation parameters.

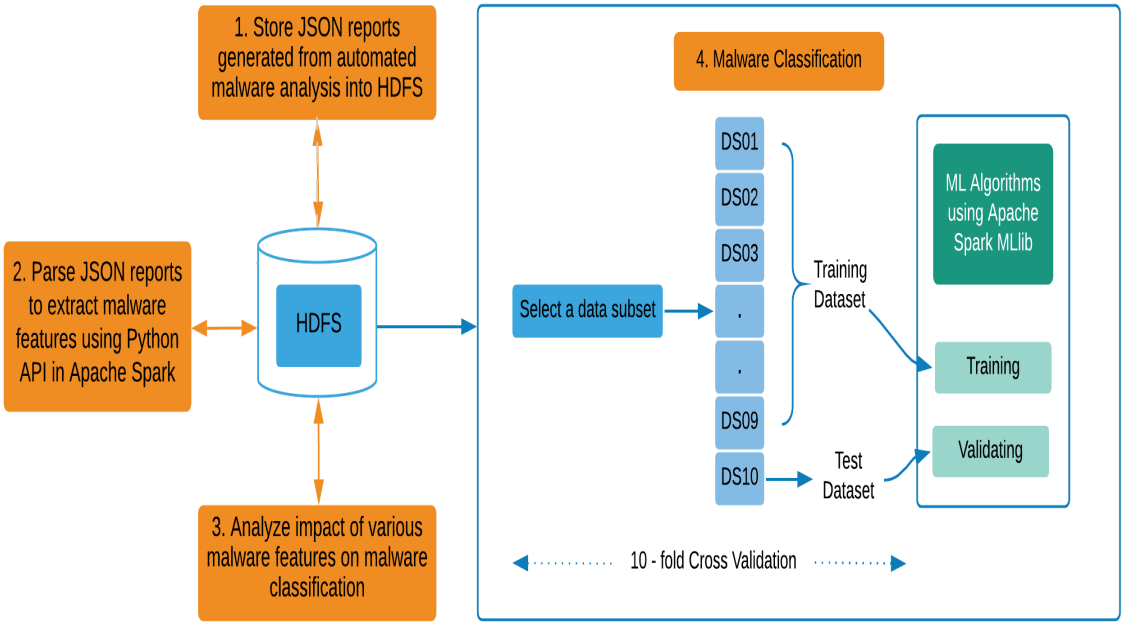


Figure 5.2: Big data framework for malware classification.

The classification results can be shared with the stakeholders to make early detection of zero-day malware. Such results can also help the security analysts to come out with preventive measures to handle such threats in future.

5.4 Malware Feature Impact Analysis

This section examines the effectiveness of features used for classification of malware and their detection. In order to understand the dataset and analyze the features used for malware classification, we build a classification model using logistic regression. It

provides a list of malware features along with their weights. Table 5.2 shows the malware features along with the corresponding weights.

Table 5.2: Malware features' dimensions with corresponding weight.

Feature	Dimension	Weight
Registry activity	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ Windows\CurrentVersion\Run%	3.1
Section's Information	One section has entropy more than 6.5	2.7
Mutex creation	Opens a mutex ShimCacheMutex	2.3
File Activity	C:\Documents and Settings\LocalService\Local Settings\Temporary Internet Files\	2.1
API Calls	API call sequence contains a sub-sequence LoadLibraryA, GetProcessAddress, SetWindowsHookExA	1.9

Relying alone on weight factor is not enough to understand the importance of features for classification purpose, because there is a possibility that a feature has acquired a high value, but may be a constant throughout the dataset. Such features are not able to discriminate the malware from the clean files. Moreover, in order to study the relevance of a feature, we considered the contribution of low-level dimensions of a feature and thus used the ranker method shown in equation (5.3) [132].

$$\text{Score Variation} = \sqrt{\text{Variance}\left[\sum_{i \in s} x_i w_i\right]} \qquad \ldots (5.3)$$

where x_i is the number of instances and w_i is the weight of i^{th} dimension of a feature (s). The relative importance of features so obtained is shown in Figure 5.3.

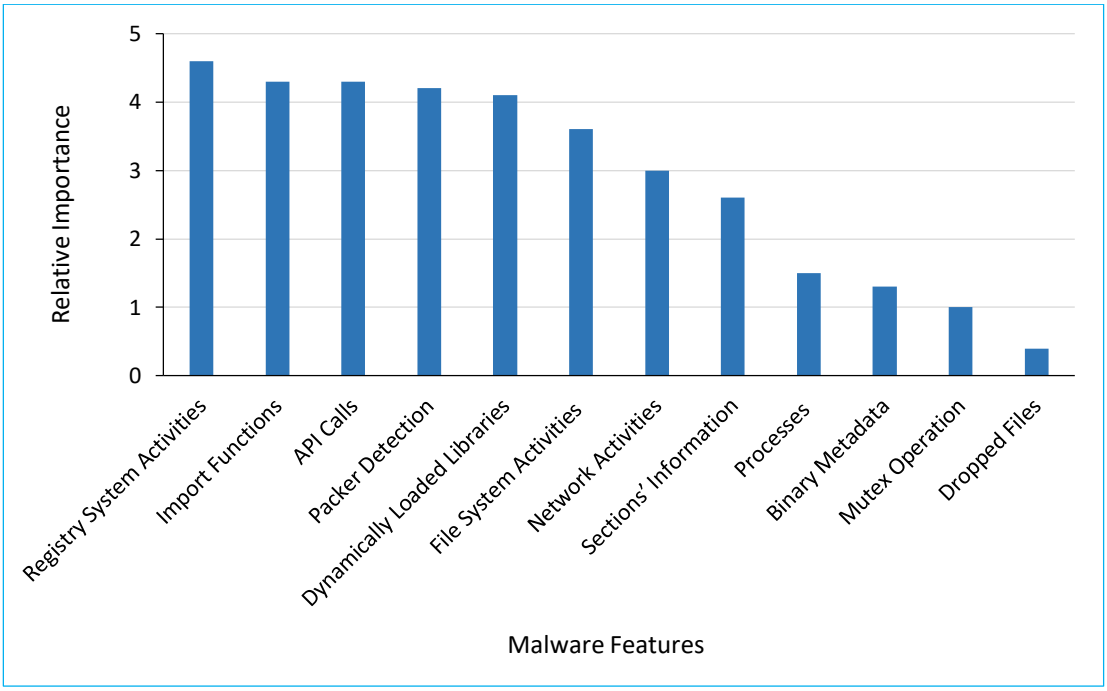


Figure 5.3: Relative importance of malware features.

5.5 Evaluation

The recent approach being used for malware detection and classification is data mining and machine learning. Machine learning algorithms can classify the behavior of a file as either malicious or benign on the basis of features collected using static or dynamic analysis of the file. The classification model built by training with labelled dataset can classify new data quickly by applying different machine learning algorithms. Thus it has been possible to build a malware detection system which has the ability to classify new malware based on the attributes obtained after performing static and dynamic malware analysis.

5.5.1 Dataset

The proposed framework for malware detection is evaluated on a dataset of 2,00,000 files including 50,000 clean files and 1,50,000 malware samples targeting Windows operating system. Thereafter, an integrated set of static (file metadata, file size, packer detection, and sections' information) and dynamic attributes (DLLs, dropped files, Windows API calls, mutex operations, file activities, registry activities, network activities, and process activities) is considered to create a combined feature vector of dimensionality 1269. This feature vector acts as a raw data to proposed big data framework. The malware data

preparation and feature extraction processes have already been described in Sections 4.2.1 and 5.3.1 respectively.

According to the scanning results of a free anti-virus, AVG (www.avg.com), the dataset covers more than 3000 malware families. The top 15 families of malware along with their number in the dataset is shown in Figure 5.4.

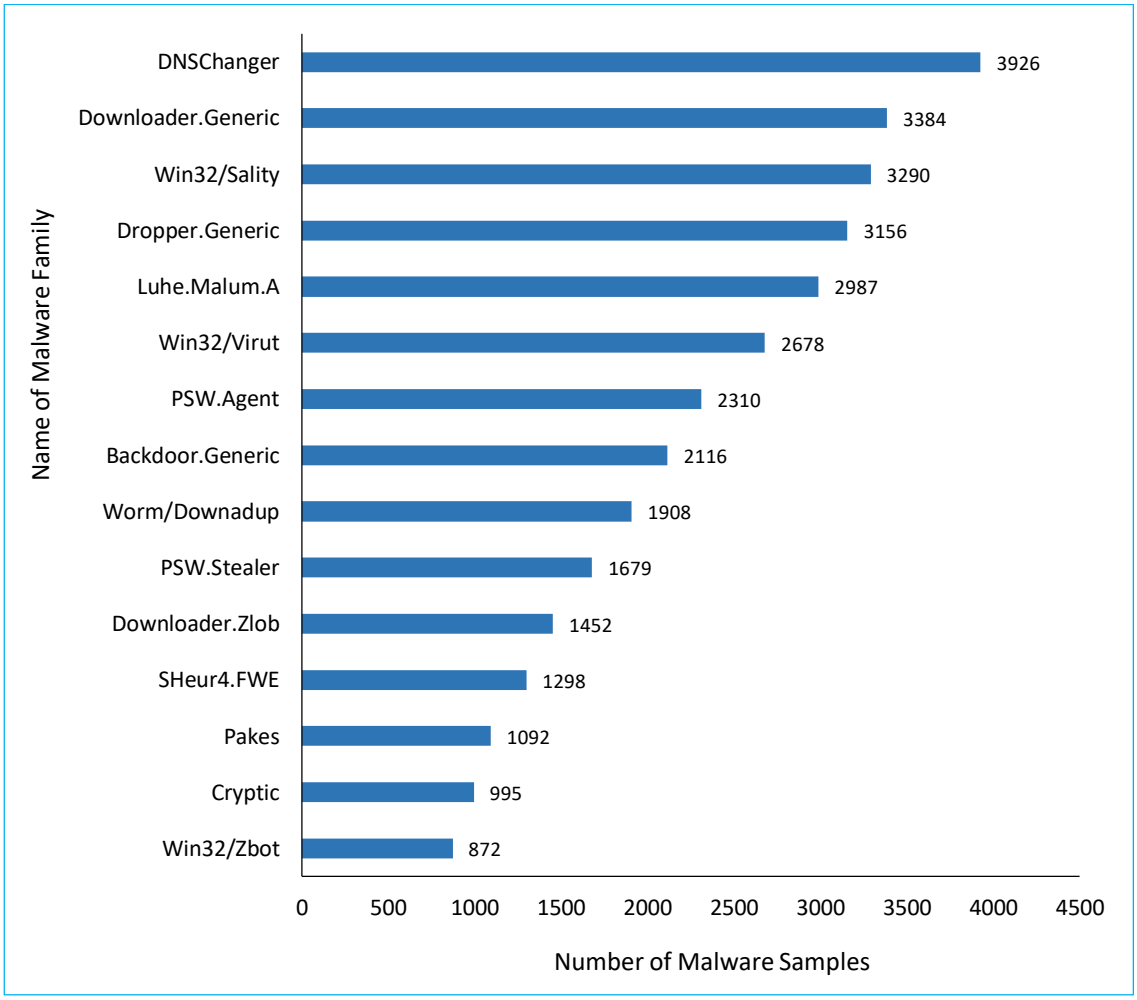


Figure 5.4: Top 15 malware families in dataset.

The number of malware samples submitted to VirusTotal for analysis during the years 2010 to 2016 are shown in Figure 5.5.

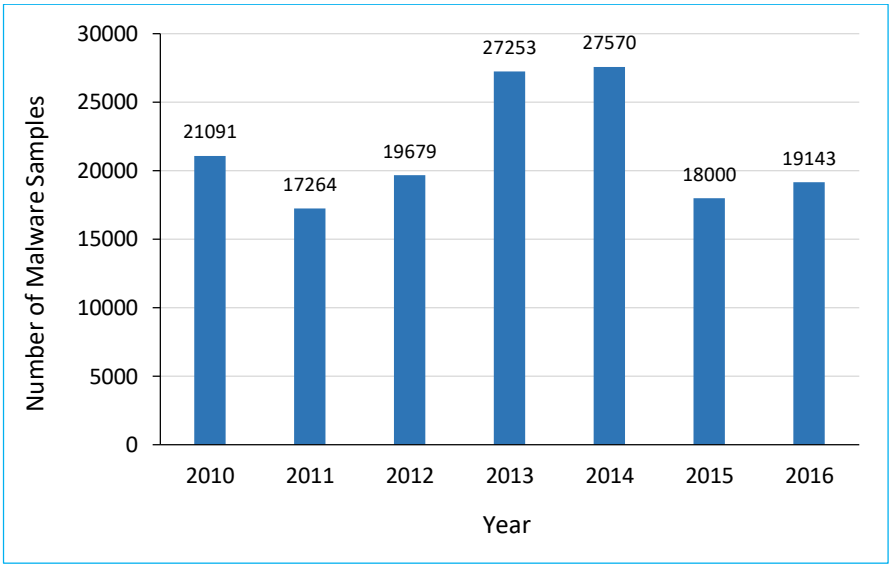


Figure 5.5: Number of malware samples (in dataset) submitted to VirusTotal in the years between 2010 and 2016.

5.5.2 Experimental Setup

The experiments are performed on Apache Spark cluster of five machines – one master and four slave machines. In all machines, we have installed Ubuntu 16.04.2 as operating system. Table 5.3 provides the details of experimental environment.

Table 5.3: Experimental environment.

Category	Description	
Software Tools	Apache Spark	Version: 2.1.0
	MLlib	
	Apache YARN	Version: 2.7.3
	Apache HDFS	
	Python	Version: 3.6.0
	Ubuntu	Version: 16.04.2
Hardware Configuration	Master: Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD	
	Slaves (04): Intel(R) Core TM i3-4160 CPU @3.60 GHz, 8 GB DDR3 1600 MHz, and 1 TB HDD	
Dataset	Malware samples: 1,70,000	
	Benign samples: 50,000	
Evaluation Parameters	10-fold cross validation	
	TPR, FPR, FNR, Accuracy, Precision	

5.5.3 Evaluation Parameters

Evaluating the performance of the machine learning techniques is a very important aspect of machine learning. It helps to provide a better understanding of the technique which results in the refinement of the parameters in the iterative process of learning. Evaluation provides the idea how well the model will work in future [165]. The result for each data element or point can be assigned any of the four categories which form the building blocks of evaluation metrics. These are known as True Positive (TP), False Negative (FN), False Positive (FP) and True Negative (TN). These outcomes can be devised in a 2x2 confusion or error matrix (see Table 5.4) and are defined in Table 5.5. In confusion matrix, each row and column denote instance of an actual and predicted class respectively, and vice versa.

Table 5.4: Table of confusion or confusion matrix for malware class.

		Predicted Class	
		Malware	Benign
Actual Class	Malware	TP	FN
	Benign	FP	TN

Table 5.5: Confusion matrix fields along with their description.

Field	**Description**
True Positive (TP)	Number of malware samples predicted correctly as malware
True Negative (TN)	Number of benign files predicted correctly as benign
False Positive (FP)	Number of benign files predicted incorrectly as malware
False Negative (FN)	Number of malware samples incorrectly predicted as benign

The various evaluation parameters used for evaluating the models in classification problems are given as follows:

1) Recall or sensitivity (True Positive Rate alias TPR) is a measure of completeness or quantity. It denotes the rate of correctly identified malicious files.

$$\text{Recall or sensitivity or TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \ldots (5.4)$$

2) False positive rate (FPR) denotes the rate of incorrectly identified benign files.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \qquad \ldots (5.5)$$

3) False negative rate (FNR) represents the rate of incorrectly identified malicious files.

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}} = 1 - \text{TPR} \qquad \ldots (5.6)$$

4) Precision (Positive Predictive Value alias PPV) measures exactness or quality of a model. It is the proportion of true results to all positive results, i.e., it is the probability for a true result to be classified correctly.

$$\text{Precision or PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \ldots (5.7)$$

5) Accuracy measures goodness of a classification model. It is proportion of correctly predicted observations (both benign and malicious) to the total observations.

$$\text{Accuracy } (\%) = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} * 100 \qquad \ldots (5.8)$$

6) F1-score is the harmonic mean of precision and recall. It lies between 0 and 1, and provides a simple way to compare classifiers. It prefers classifiers having similar precision and recall. F1-score will be high if both precision and recall are high.

$$\text{F1-score} = \frac{2 * \text{ Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 * \text{TP}}{2 * \text{TP} + \text{FN} + \text{FP}} \qquad \ldots (5.9)$$

7) Matthews Correlation Coefficient (MCC): It is used to measure and compare the performance of machine learning algorithms for binary classification [172]. It measures the correlation between predicted and the actual data labels. It takes values between -1 (perfect anti-correlation between predicted and real instances) and +1

(perfect prediction). A value equal to 0 means no better than random classification. It is computed by using equation (5.10).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad \ldots (5.10)$$

5.5.4 Cross Validation

The quality of the training data along with machine learning algorithms contribute to the success of a classification model. The simple train and test method has a couple of main limitations. First, it reduces the number of training samples which adversely affects the performance of a classifier. Second, the error rate may be misleading if splitting of training and testing sets does not represent each class proportionally. Some of the methods used to learn the classifier are random sub-sampling, leave-one-out, and cross-validation. A common practice in machine learning and deep learning is to empirically validate the performance generalization by using cross-validation mechanism [173, 174]. We have used 10-fold cross validation which partitions the training dataset randomly into 10 parts with same size. Out of these, nine subsets are used for model training and one is used for data validation. The whole process is repeated 10 times, which means that there will be ten different models with partly overlapping training sets and non-overlapping validation sets. Finally, the performance of cross-validation is calculated by taking arithmetic mean of ten performance estimates from the validation sets and is represent by equation (5.11).

$$Performance = \frac{1}{10}\left(\sum_{i=1}^{10} Performance^{i}\right) \quad \ldots (5.11)$$

5.5.5 Experimental Results

Table 5.6 provides the experimental results for predicting malware and benign classes using 10-fold cross validation. It shows that for NB classifier, the weighted average values of FPR and FNR are the highest as compared to SVM and RF. For RF classifier, the weighted average values of FPR and FNR are the lowest. RF gives the highest classification accuracy, i.e., 98.8%.

Table 5.6: Classification results using 10-fold cross validation.

Base Classifier	TPR	FPR	FNR	Accuracy (%)
NB	0.891	0.047	0.109	89.13
SVM	0.94	0.03	0.06	94.03
RF	0.989	0.011	0.011	98.88

Figure 5.6 and 5.7 visualize FPR / FNR and accuracy (%) of malware classification for all three classifiers. The results show that out of three classifiers, RF classifier is the best suited to the dataset for malware classification followed by SVM and NB respectively.

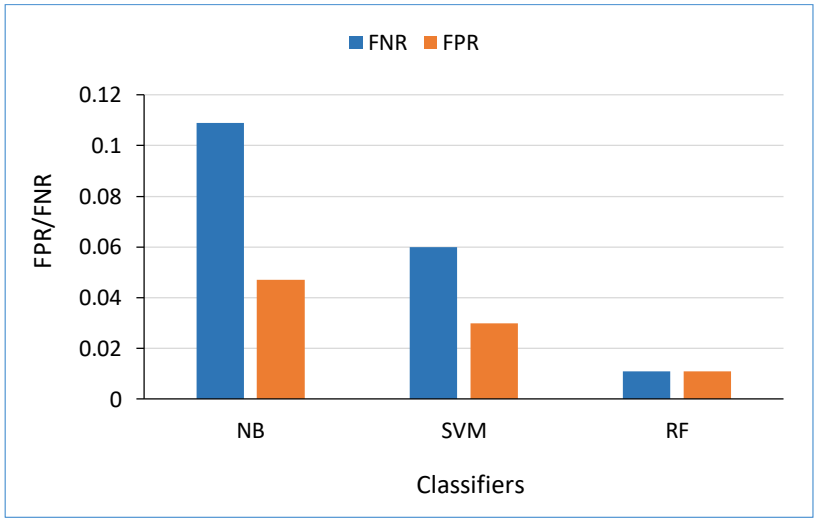


Figure 5.6: FPR/FNR comparison of classifiers.

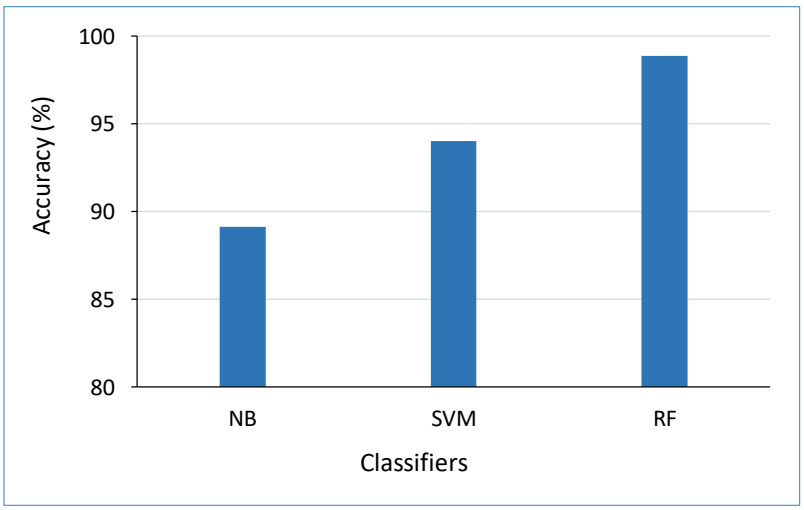


Figure 5.7: Accuracy comparison of classifiers.

5.6 Discussion

The malware samples are growing continuously at astounding rate and their detection has already been recognised as a big data problem. It is in the interest of any security organization to have scalable systems to analyze and detect malware binaries at large scale so that the proactive actions could be taken to deal with the changing threat landscape. With this goal, a framework has been proposed which is built on the top of Apache Spark using its scalable machine learning library for detecting zero-day malware binaries.

The JSON reports generated after performing automated malware analysis are stored in HDFS. Afterwards, the malware features are extracted from these reports and their impact is studied on the classification in order to understand their relative importance. The proposed scalable platform is tested and evaluated for detecting zero-day malware using a dataset covering a large number of malware families over 7 years period (from year 2010 to 2016) of submission to VirusTotal. Three machine learning algorithms, namely, NB, SVM and RF are used using Apache Spark MLlib library. The experimental results show that RF gives the best accuracy of 98.88% along with the minimum FPR and FNR followed by SVM and NB which gives an accuracy of 94.03% and 89.13% respectively.

5.7 Summary

Big data analytics and machine learning can easily analyze the data at large scale leading to detection of the potential attacks and marked increase in threat detection. This chapter proposed a scalable architecture for detecting malware binaries on the top of Apache Spark and HDFS. It makes use of three machine learning algorithms, namely, NB, SVM and RF from Apache Spark MLlib library. The results of these three classifiers are compared and it is found that RF gives the best accuracy for malware detection.

Chapter 6: Improving Malware Detection using Big Data and Ensemble Learning

Malware detection and classification is an important application of machine learning for last so many years, yet it is very difficult to point out a single model which addresses all concerns pertaining to malware detection. Although, many machine learning models have been used in detection and classification of malicious binaries, however, the performance of ensemble methods has not been investigated extensively. Sometimes, a single machine learning model or optimization of parameters does not yield the best inferences. In such scenario, a better approach is to combine multiple learning models to produce a strong model. In addition, the massive volume of malware as seen in recent years has established malware as a big data problem forcing security researchers and practitioners to deploy big data technologies to manage, store, analyze, and visualize malware data.

This chapter presents the proposed methods based on ensemble learning using big data architecture for improving the performance of malware detection. The first method is based on the weighted voting strategy of ensemble learning and the second one selects an optimal set of base classifiers for stacking purpose. The proposed solution is tested and evaluated on a dataset consisting of integrated set of features obtained after performing static and dynamic malware analysis. The experimental results show that the proposed methods improve the generalization performance in detecting new malware as compared to traditional ensemble methods. This chapter partly accomplishes objectives 2 and 3.

6.1 Ensemble Learning – An Overview

In machine learning, a united decision system is widely used to build the learning models. The ensemble learning systems unite the decisions of individual classifiers to achieve accuracy and better generalization performance. An ensemble method delivers highly

accurate classifiers by combining less accurate ones. The ensemble methods accomplish it by applying different approaches like resampling the training set, using heterogeneous algorithms, using homogeneous algorithms with diverse parameters or using different methods for combining the decisions of classifiers etc. Many ensemble methods have been suggested in various applications in the past few years and are generally categorized into following four types depending on the way these are build up [175].

6.1.1 Data Level

This type of ensemble methods (e.g. bagging methods) create subsets of training samples by using resampling methods and exploit these to build base classifiers. The results of these base classifiers are combined together using various methods. This technique works well for unstable classifiers.

6.1.2 Feature Level

Such methods can be used in two ways. The first approach builds classification models using individual feature view and then their decisions are combined together through some ensemble strategy. The second approach merges different feature views to build a classification model which gives better generalized performance as compared to the individual feature views.

6.1.3 Classifier Level

It uses either heterogeneous classifiers or homogenous classifiers with different parameters like injecting randomness into the same classifiers. Thereafter, it combines the results using some rules to overcome the bias and improve the performance of classifiers. Such models can be built by using sequential or parallel method. Boosting method falls under this category.

6.1.4 Combination Level

This method is based on combining the various classifier decisions. It includes stacking, voting, and ensemble selection as described below.

6.1.4.1 Stacking

Stacking [176] is a technique in which all base models are stacked together one over another, passing the output from lower layered model to the model above it. Stacking

helps to reduce bias or variance error depending upon the learning algorithm used. It deploys a group of homogenous or heterogeneous base classifiers, whose predictions are exploited to train a meta-classifier that gives the final prediction. The meta-classifier rectifies any errors made by the base classifiers which improves the generalized performance.

6.1.4.2 Voting

Voting is a simple combination method to combine the outputs of the basic classifiers. It allows each base classifier to predict the class of each instance of the dataset and subsequently, the class label of each instance is decided through voting scheme [175]. In majority or plurality voting, the class label $\hat{y}$ of each instance x is predicted using the vote of each base model C_j as shown in equation (6.1).

$$\hat{y} = mode\{C_1(x), C_2(x) \ldots . C_m(x)\} \qquad \ldots (6.1)$$

For example, suppose we have three classifiers C_1, C_2, C_3 that classify an instance of the dataset into binary class (either 1 or 0). According to majority voting, the label for that instance would be 1 if at least two of classifiers predict the class of the instance as 1. On the other hand, in weighted majority voting method, we can associate a weight w_j with each classifier C_j. In this case, the label $\hat{y}$ of an instance can be computed using equation (6.2).

$$\hat{y} = \arg\max \sum_{j=1}^{m} w_j f_A\big(C_j(x) = i\big) \qquad \ldots (6.2)$$

where f_A is characteristic function $[C_j(x) = i \in \mathrm{L}]$ and L is the set of unique class labels. For example, for the previous example, if the classifiers have weights as {0.6, 0.2, 0.2}, it would yield a prediction $\hat{y}$=1.

6.1.4.3 Ensemble Selection

Most of the algorithms based on ensemble learning combine all base classifiers to build an ensemble, which may sometimes diminish the performance. Ensemble selection ensures to pick up an optimal subset of accessible classifiers which improves

generalization performance. There are two steps to achieve this. First, a set of base classifiers (homogenous/heterogeneous) are build and second, a heuristic algorithm is employed to choose an optimal set of base classifiers.

6.2 Proposed Methodology

It is clear from the literature that multiple feature view helps in providing an inclusive depiction of a program and thus helps in detecting the malware more accurately. So, the proposed methodology makes use of multiple features obtained through static and dynamic malware analysis. After extracting the features, we have used five diverse inducers as base classifiers to build models which are evaluated on the parameters like true positive rate, true negative rate, precision, accuracy, F-measure and MCC computed using equations (5.4) to (5.10) and 10-fold cross validation using equation (5.11) as explained in section 5.5.3 and 5.5.4 respectively. The results obtained from these are used to design the schemes for:

- Computing the weights for base classifiers to be used for weighted voting ensemble method.
- Selecting an optimal pool of classifiers to be used for stacking.

The results obtained from the proposed methods are compared with the best base classifiers and some of the traditional ensemble methods. Figure 6.1 presents an overview of proposed methodology and its various steps are discussed below:

6.2.1 Malware Classification Using Base Classifiers

We used five diverse inducers as base classifiers, namely, NB, KNN [177], Decision Table (DT) [178], SVM, and RF. Each of these classifiers belongs to different family of classifiers and thus uses different approach to classify the input data. These are used for building base models and are evaluated on the basis of various evaluation parameters. The NB, SVM, and RF are explained in Section 5.2, and the other two base classifiers are described below:

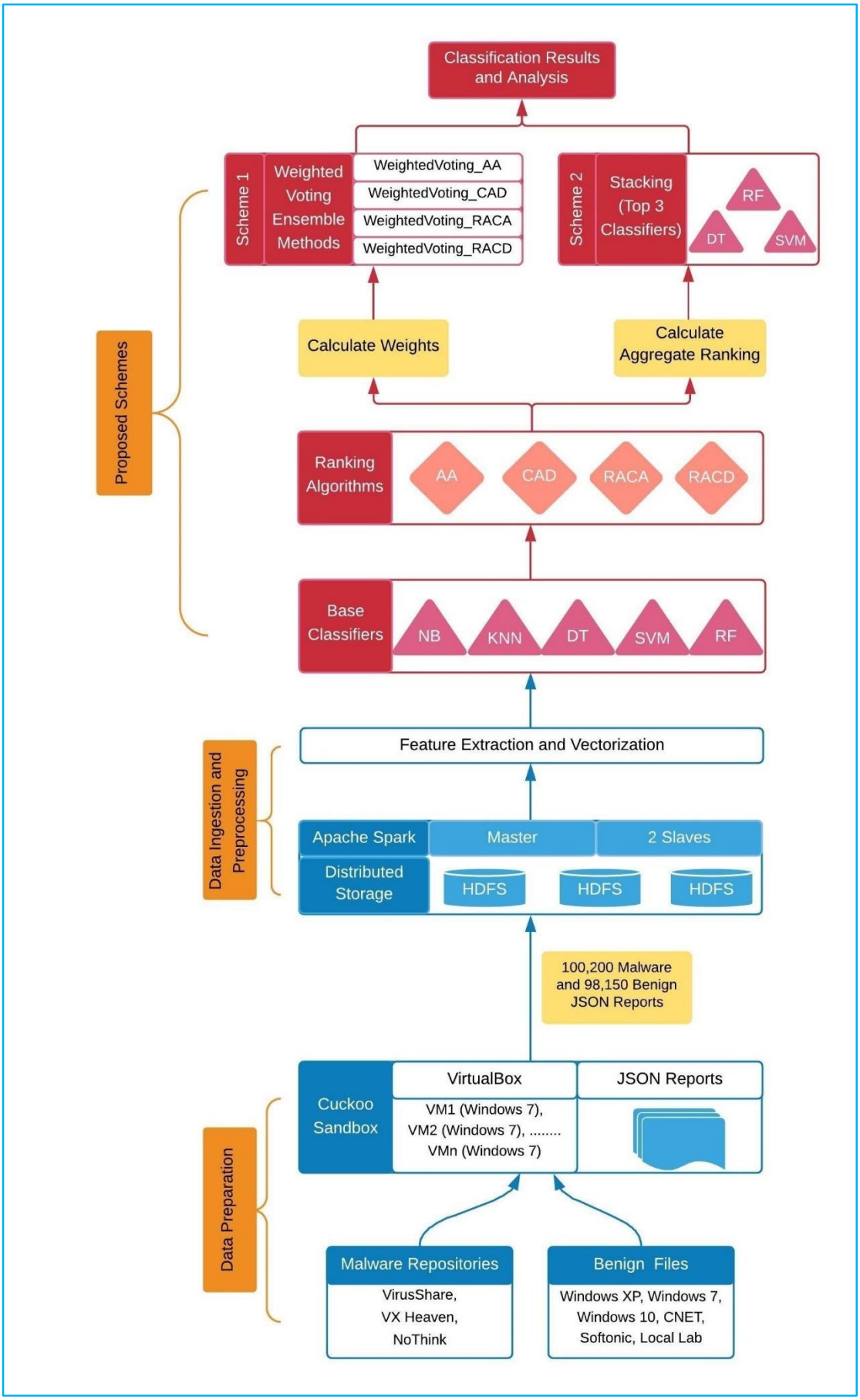


Figure 6.1: Proposed methodology for detecting zero day malware.

6.2.1.1 K-Nearest Neighbour

KNN is a representative of lazy machine learning algorithms that simply stores the training data and waits until it is provided with a test data [177]. On getting the test data, it builds the classification model based on the matching to the already available training data instances. However, these learners support incremental learning, but are computationally expensive. We have used k = 5 and normalized Euclidean distance to find the training instance closest to the given test instance. Euclidean distance between two points having coordinates (x, y) and (a, b) is calculated using equation (6.3).

$$Dist\big((x,y),(a,b)\big) = \sqrt{(x-a)^2 + (y-b)^2} \qquad \ldots (6.3)$$

If multiple instances are there at the smallest distance from the test instance, the first such instance is used as its class.

6.2.1.2 Decision Table

It is a type of classifier in which specific features are selected while performing learning process [178, 179]. It is done by computing cross validation performance of the table for different sub-sets of attributes and selecting the sub-set which is the best performer. The cross-validation error from a DT is obtained by manipulating the class counts related to each entry of the table, as the structure of the table does not change when instances are added or deleted. The feature space is usually searched by using best-first search algorithm.

6.2.2 Proposed Schemes

We have proposed following two schemes for building an ensemble classifier based on: (1) weighted voting, and (2) selecting an optimal pool of classifiers to be used for stacking. These schemes are based on the rank algorithms designed in [143].

6.2.2.1 Scheme I

It has been observed from the literature that most of the classifiers usually give different results for different classes. For example, they do not classify the malware and benign classes with equal accuracy. This fact has been exploited to compute the weights of base

classifiers which can be used for implementing weighted voting. The proposed approaches for computing the ranks and weights are described below:

a) Average Accuracy (AA)

In this method, the ranking is computed by taking the average of prediction accuracies of classes. The base classifier with better average accuracy will get higher rank. Consider that there are c classifiers denoted by $x_{1,}x_{2,}$ ..... x_c being used to classify the data into two classes, i.e., malware and benign. Let P_{m,x_i} and P_{b,x_i} are the accuracies of malware and benign class for a classifier x_i respectively. The average accuracy P_{avgx_i} of each classifier is computed by using equation (6.4) as given below:

$$P_{avgx_i} = \frac{n_m \times P_{m,x_i} + n_b \times P_{b,x_i}}{n_m + n_b} \quad \forall \text{ I} \in \{1, 2, \ldots \text{c}\} \qquad \ldots(6.4)$$

Where n_m and n_b denote number of malicious and benign files respectively. Suppose A= $\{P_{avgx_1}, P_{avgx_2} \ldots P_{avgx_c}\}$ is the set of average accuracies of all classifiers. Then the rank is calculated by using $Rank_{desc}(\,)$ function which assigns the rank to every classifier on the basis of its average accuracy as depicted in equation (6.5) (larger the value of P_{avgx_i}, higher is the rank).

$$Rank(x_{i,}\, P_{avgx_i}) = Rank_{desc}(A) \quad \forall \text{ i} \in \{1, 2, \ldots, \text{c}\} \qquad \ldots (6.5)$$

Then the weight of each classifier is computed by dividing the rank of that classifier by the sum of the rank values of all classifiers as shown in equation (6.6).

$$Wt_{x_i} = \frac{Rank(x_{i,}\, P_{avgx_i})}{\sum_{i=1}^{c} Rank(x_{i,}\, P_{avgx_i}))} \quad \forall \text{ i} \in \{1, 2, \ldots, \text{c}\} \qquad \ldots (6.6)$$

b) Class Accuracy Differential (CAD)

In this approach, a weight is assigned to each base classifier on the basis of average accuracy and the absolute difference between the class accuracies. Let D_{x_i} is the ratio of average performance accuracy and the difference of accuracies of malware and benign class $|P_{m,x_i} - P_{b,x_i}|$ for a classifier. It is computed using equation (6.7).

$$D_{x_i} = \frac{P_{avgx_i}}{|P_{m,x_i} - P_{b,x_i}|} \quad \forall\, i \in \{1,2,\dots,c\} \qquad \dots (6.7)$$

Suppose D= { $D_{x_1}, D_{x_2}, \dots \; D_{x_c}$} is the set of values of class differential for all classifiers. Then the rank is calculated using $Rank_{desc}(\,)$ function (equation (6.8)) which assigns the rank to every classifier on the basis of its class accuracy differential (larger the value of D_{x_i}, higher is the rank).

$$Rank\left(x_{i,}\, D_{x_i}\right) = Rank_{desc}(D) \qquad \dots (6.8)$$

Then the weight of each classifier is computed by dividing the rank of that classifier by the sum of the rank values of all classifiers (equation (6.9)).

$$Wt_{x_i} = \frac{Rank\left(x_{i,}\, D_{x_i}\right)}{\sum_{i=1}^{c} Rank\left(x_{i,}\, D_{x_i}\right)} \forall\, i\, \{1,2,\dots,c\} \qquad \dots (6.9)$$

c) Ranked Aggregate per Class Accuracies Based Approach (RACA)

This approach assigns rank individually to each classifier based on the class accuracy. The final rank for a classifier is computed using the sum of per class ranking. Let B= $\{B_{x_1}, B_{x_2}, \dots B_{x_c}\}$ be the set of accuracies of all classifiers for benign class and M= $\{M_{x_1}, M_{x_2}, \dots M_{x_c}\}$ is the set of accuracies of all classifiers for malware class. The rank to every classifier is assigned separately on the basis of class accuracies using $Rank_{desc}(\,)$ function (equations (6.10) and (6.11)) (larger the value of B_{x_i} or M_{x_i}, higher is the rank).

$$Rank(x_i, b) = \; Rank_{desc}(B) \qquad \dots (6.10)$$

$$Rank(x_i, m) = \; Rank_{desc}(M) \qquad \dots (6.11)$$

Now compute the aggregate of per class rank as shown in equation (6.12).

$$Z_{x_i} = Rank(x_i, b) + Rank(x_i, m) \quad \forall \text{ i } \in \{1, 2, \dots, c\} \qquad \dots (6.12)$$

Let Z= $\{Z_{x_1}, Z_{x_2} \ldots Z_{x_c}\}$ is the set of values of aggregate per class rank for all classifiers. Final rank for the classifier is computed using ranking function, i.e., $Rank_{desc}(\,)$ as shown in equation (6.13).

$$Rank(x_{i,}\ Z_{x_i}) = Rank_{desc}(Z) \qquad \ldots (6.13)$$

Then the weight of each classifier is computed by dividing the rank of that classifier by the sum of the rank values of all classifiers as shown in equation (6.14).

$$Wt_{x_i} = \frac{Rank(x_{i,}\ Z_{x_i})}{\sum_{i=1}^{c} Rank(x_{i,}\ Z_{x_i})} \forall\ i\ \{1,2,\ldots,c\} \qquad \ldots (6.14)$$

d) Ranked Aggregate of Average Accuracy and Class Differential (RACD)

In this approach, the final ranking is computed on the basis of values obtained after summing up the rank obtained using average accuracy of the classifier and difference between the class accuracies. Here, let t_{x_i} is the class accuracy differential as shown in equation (6.15).

$$t_{x_i} = \left|P_{m,x_i} - P_{b,x_i}\right|\ \forall\ i\ \in \{1,2,\ldots,c\} \qquad \ldots (6.15)$$

Suppose T = $\{t_{x_1}, t_{x_2}, \ldots t_{x_c}\}$ is the set of class differential values for all classifiers. Then the rank for each class is computed using $Rank_{ascen}(\,)$ function (equation (6.16)) which assigns the rank to every classifier on the basis of t_{x_i} (smaller the value of t_{x_i}, higher is the rank).

$$Rank(x_i, t_{x_i}) = Rank_{ascen}(T) \qquad \ldots (6.16)$$

Aggregate the value of rank for both average accuracy and class differential for each classifier using equation (6.17).

$$H_{x_i} = Rank(x_{i,}\ P_{avgx_i}) + Rank(x_i, t_{x_i}) \qquad \ldots (6.17)$$

Suppose H = { H_{x_1}, H_{x_2} ... H_{x_c}} is the set of aggregate rank on the basis of average accuracy and class differential accuracies for all classifiers. The final ranking would be computed using ranking function $Rank_{desc}(\ )$ as described in equation (6.18).

$$Rank(x_i, H_{x_i}) = Rank_{desc}(H) \qquad \ldots (6.18)$$

Then the weight of each classifier is computed by dividing the rank of that classifier by the sum of the rank values of all classifiers as shown in equation (6.19).

$$Wt_{x_i} = \frac{Rank(x_i, H_{x_i})}{\sum_{i=1}^{c} Rank(x_i, H_{x_i})} \forall\ i\ \{1,2, \ldots, c\} \qquad \ldots (6.19)$$

6.2.2.2 Scheme II

We have further aggregated the rank assigned to various classifiers by different ranking algorithms to select an optimal set of base classifiers for implementing stacking. The top three base classifiers (i.e. 60% of base classifiers) are selected as an optimal set of base classifiers instead of considering all the base classifiers.

6.3 Evaluation

This section describes the outcomes of proposed solution built which is implemented using Apache Spark. An integrated set of selected features (as described in Section 5.3.1) are combined with the proposed ensemble methods and the results are compared with best base classifier and traditional ensemble methods. In evaluation, we noted the model performance in terms of precision, recall, accuracy, and F1-score. These are discussed in detail in the following sub-sections:

6.3.1 Dataset

The proposed ensemble methods for malware detection are evaluated on a dataset of 1,98,350 files including 1,00,200 malicious and 98,150 benign files. Thereafter, an integrated set of static (file metadata, file size, packer detection, and sections' information) and dynamic attributes (DLLs, dropped files, Windows API calls, mutex operations, file activities, registry activities, network activities, and process activities) is considered to create a combined feature vector of dimensionality 1269. This feature vector acts as a raw

data to proposed big data framework. The malware data preparation and feature extraction processes have already been described in Sections 4.2.1 and 5.3.1 respectively.

6.3.2 Experimental Environment

This section discusses the experimental setup in context to big data. The proposed architecture leverage the open-source components like Apache Spark for efficient preprocessing of data, HDFS for distributed storage, and Python for data-intensive analysis which are already discussed in Sections 1.4.2 and 4.2.2. The experiments are conducted on Apache Spark based multinode cluster - a master and two worker nodes, with Ubuntu 18.04.2 as operating system. The master node has Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD. The worker nodes are Intel(R) Core TM i5-8250U CPU @1.6 GHz, 8 GB DDR4, and 1 TB HDD. Table 6.1 provides the details of experimental environment.

Table 6.1: Experimental environment.

Category	**Description**	
Software Tools	Apache Spark	Version: 2.4.0
	Apache HDFS	Version: 3.1.1
	Python	Version: 3.6.0
	Ubuntu	Version: 18.04.2
Hardware Configuration	Master: Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD	
	Slaves (02): Intel(R) Core TM i5-8250U CPU @1.6 GHz, 8 GB DDR4, and 1 TB HDD	
Dataset	Malware samples: 1,00,200	
	Benign samples: 98,150	
Evaluation Parameters	10-fold cross validation	
	TPR, FPR, FNR, Accuracy, Precision, F-measure, MCC	

6.3.3 Evaluation Results

We start examining the experimental results using big data environment for classification of dataset. First of all, five base models are build using 10-fold cross validation whose performance is calculated by using equation (5.11). Thereafter, the various evaluation parameters like TPR, FPR, FNR, precision, accuracy (%), F-measure, and MCC (as

described in Section 5.5.3) are computed using equations (5.4) to (5.10). Table 6.2 depicts the classification results of base classifiers.

Table 6.2: Classification results using base classifiers.

Base Classifier	TPR	FPR	FNR	Precision	F-Measure	Accuracy (%)	MCC
NB	0.844	0.159	0.156	0.844	0.844	84.2	0.685
KNN	0.921	0.067	0.079	0.933	0.927	92.6	0.854
DT	0.944	0.045	0.056	0.955	0.949	94.9	0.899
SVM	0.896	0.108	0.104	0.894	0.894	89.3	0.788
RF	0.977	0.015	0.023	0.985	0.981	98.1	0.961

Figures 6.2, 6.3, and 6.4 visualize the comparison of base classifiers on the basis of FPR/FNR, Accuracy, and MCC respectively. Figure 6.2 depicts that the values of FPR and FNR are maximum and minimum for NB and RF classifier respectively. Further, RF has delivered highest average accuracy, i.e., 98.1% followed by 94.9% for DT (Figure 6.3). Figure 6.4 shows that RF records the maximum value using MCC, i.e., 0.961 (depicting a highest correlation between predicted and the actual data labels) followed by DT, i.e., 0.899.

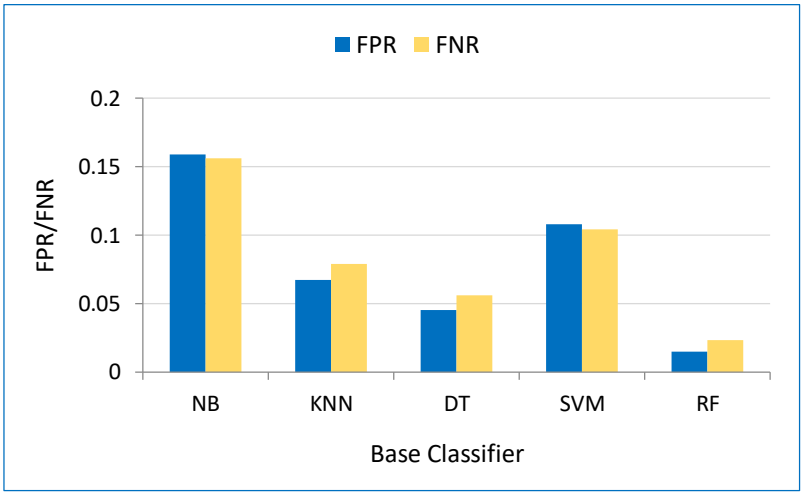


Figure 6.2: Comparison of base classifiers on the basis of FPR/FNR.

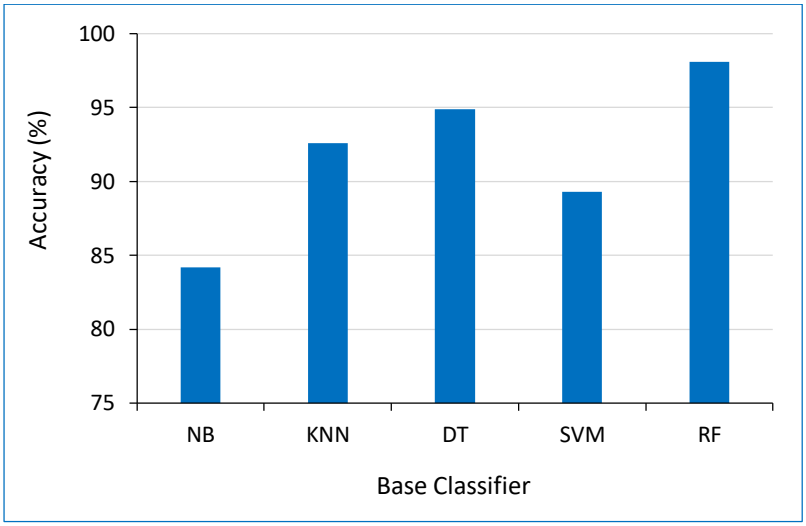


Figure 6.3: Comparison of base classifiers on the basis of accuracy.

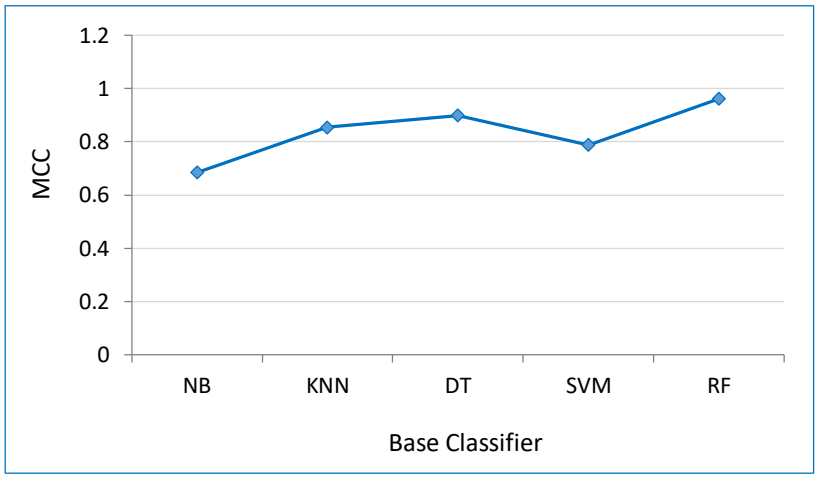


Figure 6.4: Comparison of base classifiers on the basis of MCC.

In order to obtain the rank and the weight of each classifier, the four methods are applied on the classification results obtained for base classifiers. All computations are based on class accuracies, i.e., TPR (accuracy of malicious class) and TNR (accuracy of benign class). Table 6.3 shows the class accuracies for each of the classifier obtained from 10-fold cross validation method, and also presents the ranking and weights calculated using AA, CAD, RACA and RACD methods.

Table 6.3: Rank and weight of different classifiers using ranking algorithms.

Classifier	TPR	TNR	AA		CAD		RACA		RACD	
			Rank	Weight	Rank	Weight	Rank	Weight	Rank	Weight
NB	0.844	0.841	1	0.067	5	0.300	2	0.133	4	0.2
KNN	0.921	0.933	3	0.200	1	0.067	3	0.200	3	0.15
DT	0.944	0.955	4	0.267	2	0.133	4	0.267	4	0.2
SVM	0.896	0.892	2	0.133	4	0.267	1	0.067	4	0.2
RF	0.977	0.985	5	0.300	3	0.200	5	0.300	5	0.25

Each of the four ranking methods computed a different set of rank and weight for the base classifiers. The weights obtained from the different methods are used separately in an ensemble method based on weighted voting. Further, the ranks obtained from these methods are summed up to compute the aggregated rank of each classifier. Based on aggregated rank, the top three classifiers (i.e. 60% of base classifiers) are picked for stacking purpose. The traditional ensemble methods, i.e., majority voting and stacking (considering all base classifiers) are compared with the results obtained from the proposed methods. In stacking, logistic regression (LR) has been deployed as level-2 meta-classifier. LR is an algorithm which is used to classify the data instances into a discrete set of classes. It converts the output using the logistic sigmoid function (equation 6.20) and gives a probability value which can be easily mapped to distinct classes.

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 \ldots b_n x_n)}} \qquad \ldots (6.20)$$

where p is the probability generating value between 0 and 1, and $(b_0 + b_1 x_1 \ldots b_n x_n)$ is the input to the function having n variable inputs.

For mapping the probability to discrete classes, a threshold value is selected. If the value of p for a data instance is greater than or equal to selected threshold, it is classified as 1 (malware), otherwise 0 (benign). It is represented using equation (6.21).

$$If\ p \geq 0.5, class = 1, else\ class = 0 \qquad \ldots (6.21)$$

Table 6.4 shows the classification results for traditional ensemble methods (i.e. majority voting, and stacking considering all base classifiers) and proposed methods (WeightedVoting_AA, WeightedVoting_CAD, WeightedVoting_RACA and WeightedVoting_RACD, and stacking of top three classifiers).

Table 6.4: Comparison of traditional ensemble methods with the proposed ensemble methods.

Ensemble Method	**TPR**	**FPR**	**FNR**	**Precision**	**F-Measure**	**Accuracy (%)**	**MCC**
Majority Voting	0.988	0.014	0.012	0.986	0.987	98.7	0.974
WeightedVoting_AA	0.994	0.008	0.006	0.992	0.993	99.3	0.986
WeightedVoting_CAD	0.987	0.016	0.013	0.984	0.985	98.5	0.971
WeightedVoting_RACA	**0.996**	**0.005**	**0.004**	**0.995**	**0.995**	**99.5**	**0.991**
WeightedVoting_RACD	0.989	0.013	0.011	0.988	0.988	98.8	0.977
Stacking (All classifiers)	0.993	0.009	0.007	0.991	0.992	99.2	0.984
Stacking (RF, DT, SVM)	0.994	0.009	0.006	0.991	0.992	99.2	0.985

Figure 6.5 visualizes the FPR/FNR for the traditional majority voting, stacking (considering all classifiers) and the proposed methods. It vividly shows that FPR/FNR for WeightedVoting_RACA is minimum. Figure 6.6 shows the accuracy comparison of best base classifier with that of all ensemble methods, which provides better accuracy than the best base classifier, i.e. RF. Further, it shows that the proposed weighted voting based methods, except WeightedVoting_CAD, give better accuracy as compared to the majoring voting, stacking and the best base classifier. Further, it is evident that both stacking (considering all classifiers) and stacking with top 3 base classifiers give the same accuracy. Therefore, it is better to use the optimal set of classifiers instead of considering all base classifiers as it reduces the overhead in terms of computation time.

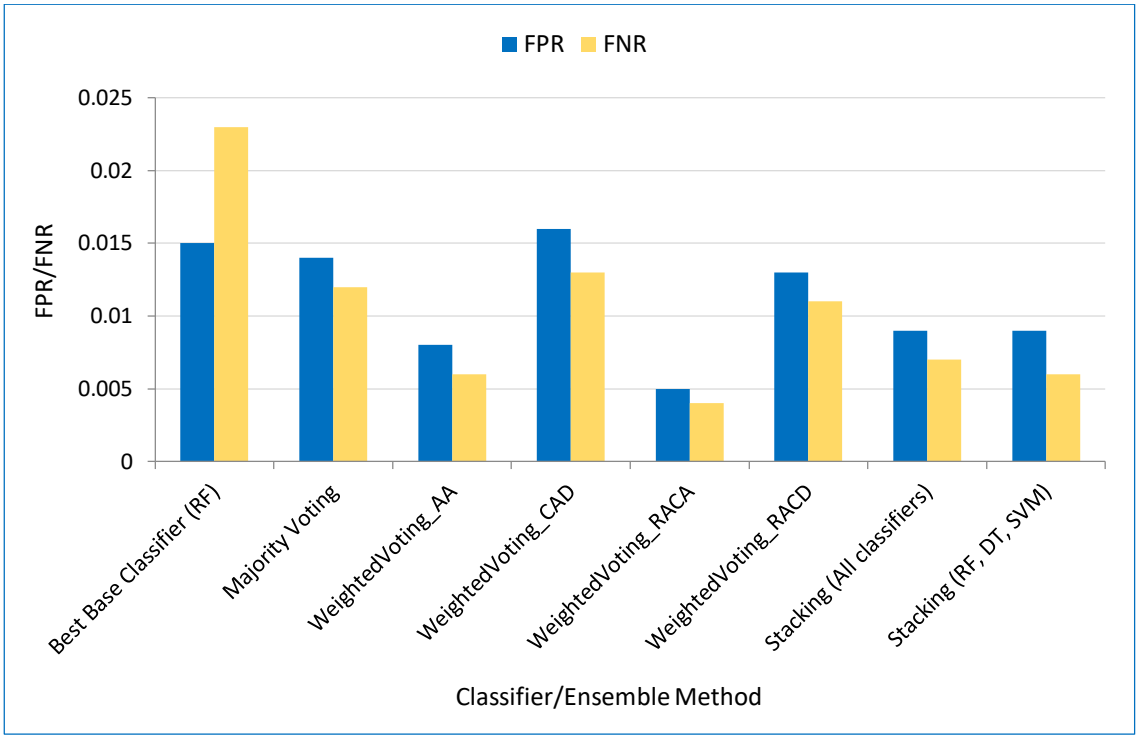


Figure 6.5: Comparison of best base classifier, traditional and proposed ensemble methods using FPR/FNR.

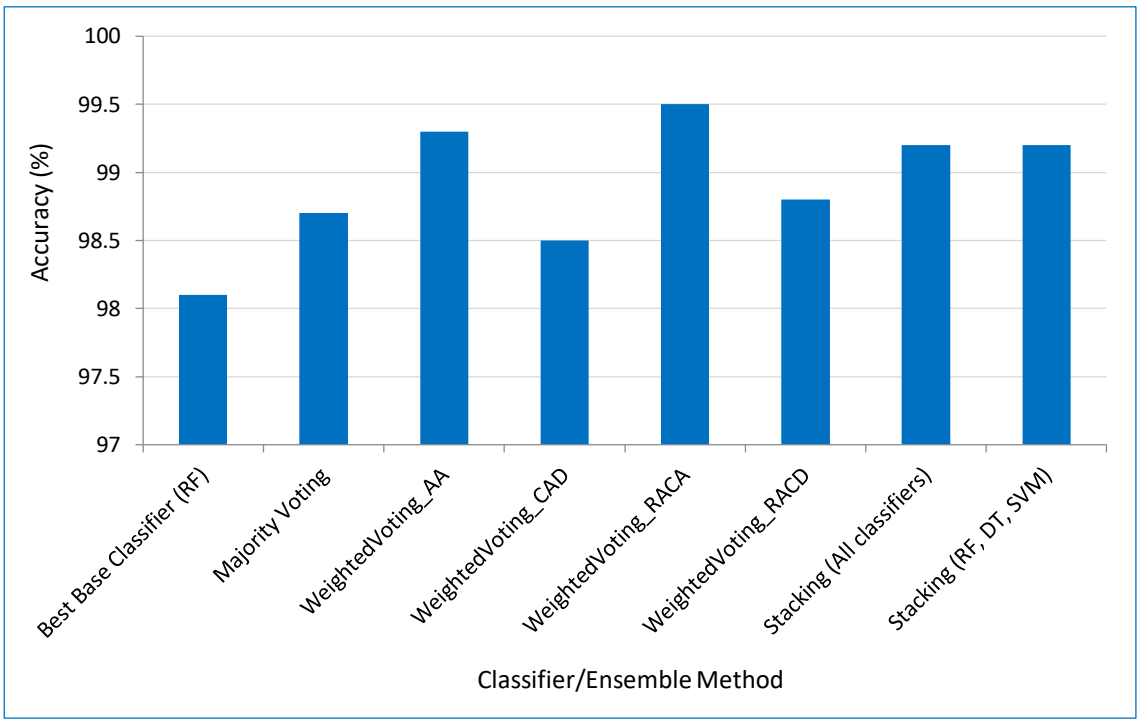


Figure 6.6: Comparison of best base classifier, tradition ensemble and proposed ensemble methods using accuracy.

Figure 6.7 visualizes MCC for comparing the performance of all the base classifiers, proposed and traditional ensemble methods for classification of unknown malware. It shows that the value of MCC is close to 1 for all ensemble methods and WeightedVoting_RACA has the highest MCC value. This means that the ensemble methods provide better correlation between predicted and actual data labels. Therefore, the proposed schemes for ensemble methods can be used to improve the generalized performance for detecting unknown malware.

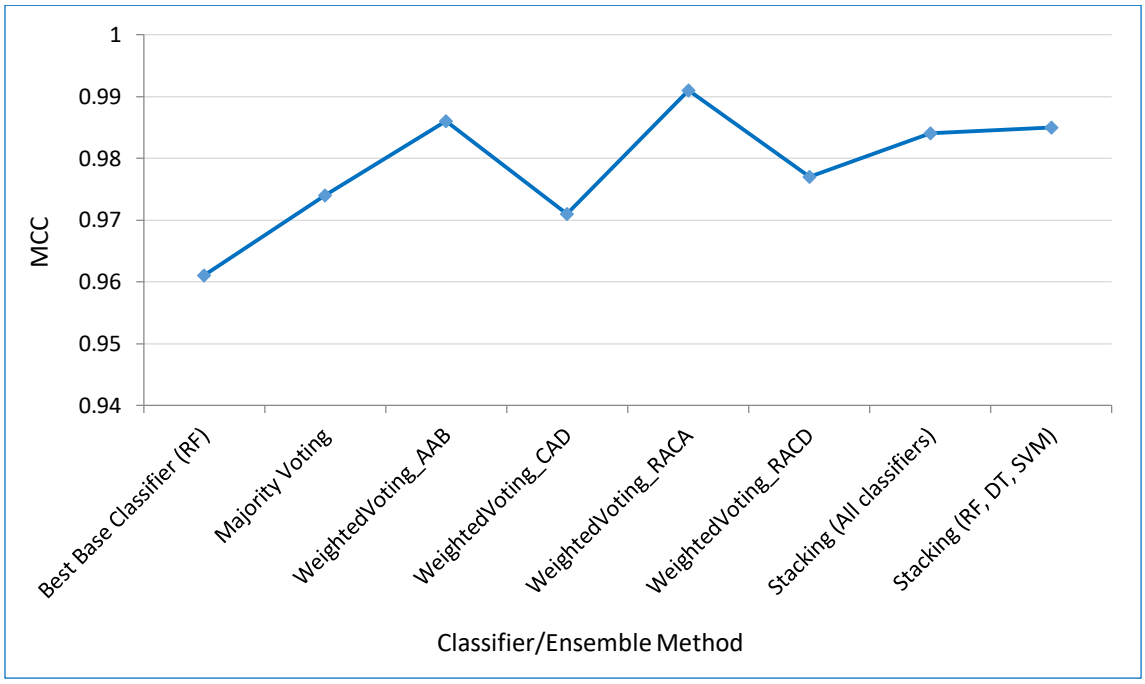


Figure 6.7: Comparison of base classifiers, tradition ensemble, and proposed ensemble methods using MCC.

6.4 Discussion

An ensemble method is highly robust while dealing with data uncertainty, though it brings more complexity and computational cost. Recently, it has become a popular technique in various competitions and is commonly used to improve accuracy benchmarks for various solutions.

In this work, we have proposed two methods based on ensemble learning and big data for improving the performance of malware detection at large scale. The first method is based on the weighted voting strategy of ensemble learning, and the second method chooses an optimal set of base classifiers for stacking purpose. The proposed methods are

implemented using Apache Spark, a popular big data processing framework, and their performance is tested and evaluated on a dataset having integrated set of static and dynamic features of 1,98,350 Windows files including 1,00,200 malicious and 98,150 benign files.

The experimental results show that the proposed approach improves the generalization performance in detecting new malware as compared to traditional ensemble methods. Further, it shows that the proposed weighted voting based methods, except WeightedVoting_CAD, give better accuracy as compared to the majoring voting, stacking and the best base classifier. Further, it is evident that both stacking (considering all classifiers) and stacking with top 3 base classifiers give the same accuracy. Therefore, it is better to use the optimal set of classifiers instead of considering all base classifiers as it reduces the overhead in terms of computation time.

6.5 Summary

This chapter presented malware classification system using an integrated set of static and dynamic features and ensemble learning on Apache Spark. Two methods are designed for building an ensemble classifier. These methods are based on: (1) weighted voting, and (2) selecting an optimal pool of classifiers used in stacking. The proposed methods are evaluated on a dataset consisting of 1,98,350 Windows files including 1,00,200 malicious and 98,150 benign files. The experimental results conclude that the proposed methods give better generalized performance for detecting new malware as compared to the base classifiers and traditional ensemble methods.

Chapter 7: Malware Classification using Big Data and Deep Neural Network

Many of the machine learning solutions suffer from high FPR and low scalability restricting their wider adoption and deployment. In recent years, deep learning, a subfield of machine learning, has resurged and reported outstanding performance in tackling many classification problems in a wide range of fields.

This chapter presents the proposed solution for malware detection by bringing together the benefits of Apache Spark – a big data processing engine, and Keras along with TensorFlow, a deep learning framework. Apache Spark is deployed for efficient preprocessing of data, and Keras along with TensorFlow is used to develop and train DNN. The resulting model is trained and evaluated on a dataset having integrated set of malware static and dynamic features. The resulting classification model gives better performance than the traditional machine learning algorithms. This chapter accomplishes objectives 2 and 3.

7.1 Introduction

In last decade, machine learning has become the forefront of research and industry security applications. It uses static and dynamic malware features to identify anomalies, correlations, trends, and patterns to discover a malicious binary. It is pertinent to understand that one of the important goal of malware classification system is to achieve extremely low FPR and a reasonably low FNR. The impact of FPR is much higher as it may cause deletion of a critical system file preventing a computer to boot properly, though it is extremely difficult to achieve low FPR as it requires huge labeled malware and benign samples to train a model. Unfortunately, many traditional machine learning based malware detection applications suffer from high FPR and FNR because of the following

reasons: (1) the limited dataset used for training the model, (2) lack of optimal feature selection used for building the classification models, (3) ever-growing complexity and variety of malicious programs (like sophisticated APTs), and (4) diverse nature of system calls made by programs [180].

As a response to abovementioned problems, many organizations are adopting big data technologies which can easily ingest, store, process and analyze the massive threat data. The big data technologies have already become cornerstones of digital transformation as these are now more accessible and cost-effective due to the availability of commoditized hardware and open source tools. Further, the security researchers are augmenting malware detection and classification with deep learning. Deep learning, an umbrella term representing recent advances in neural networks, has become extremely popular recently covering several fields. It mimics the human brain and processes the information at multiple levels of abstractions from high-dimensional raw data [181]. It uses stacked NNs allowing non-linear feature transformation to extract abstract complex patterns which are generalized to unknown malware data. The sudden resurgence in deep learning is attributed to growing volumes of data, advances in algorithms, availability of easy-to-use open-source libraries, and computing power which is now available at a reasonable cost. Nevertheless, deep learning based model learning takes more time to obtain adequate information about classification. There is an interesting trade-off in selection machine learning and deep learning methods. While the former appear to be fast, but not so accurate; the latter seems to be slow, yet yields more precise classification results.

7.2 Methodology

Malware detection is a binary classification problem which separates the elements of a given dataset into two groups - benign and malicious. In this section, we describe the proposed approach for malware detection which is developed to combine the benefits of Apache Spark – a big data processing engine, and Keras on the top of TensorFlow, a popular and widely adopted deep learning framework. While Apache Spark does efficient data preprocessing, Keras along with TensorFlow helps in developing a DNN model which provides best classification accuracy. In all, the proposed approach consists of three main steps: (1) data preparation and preprocessing, (2) feature extraction and aggregation, and (3) malware classification using DNN and traditional classifiers, namely

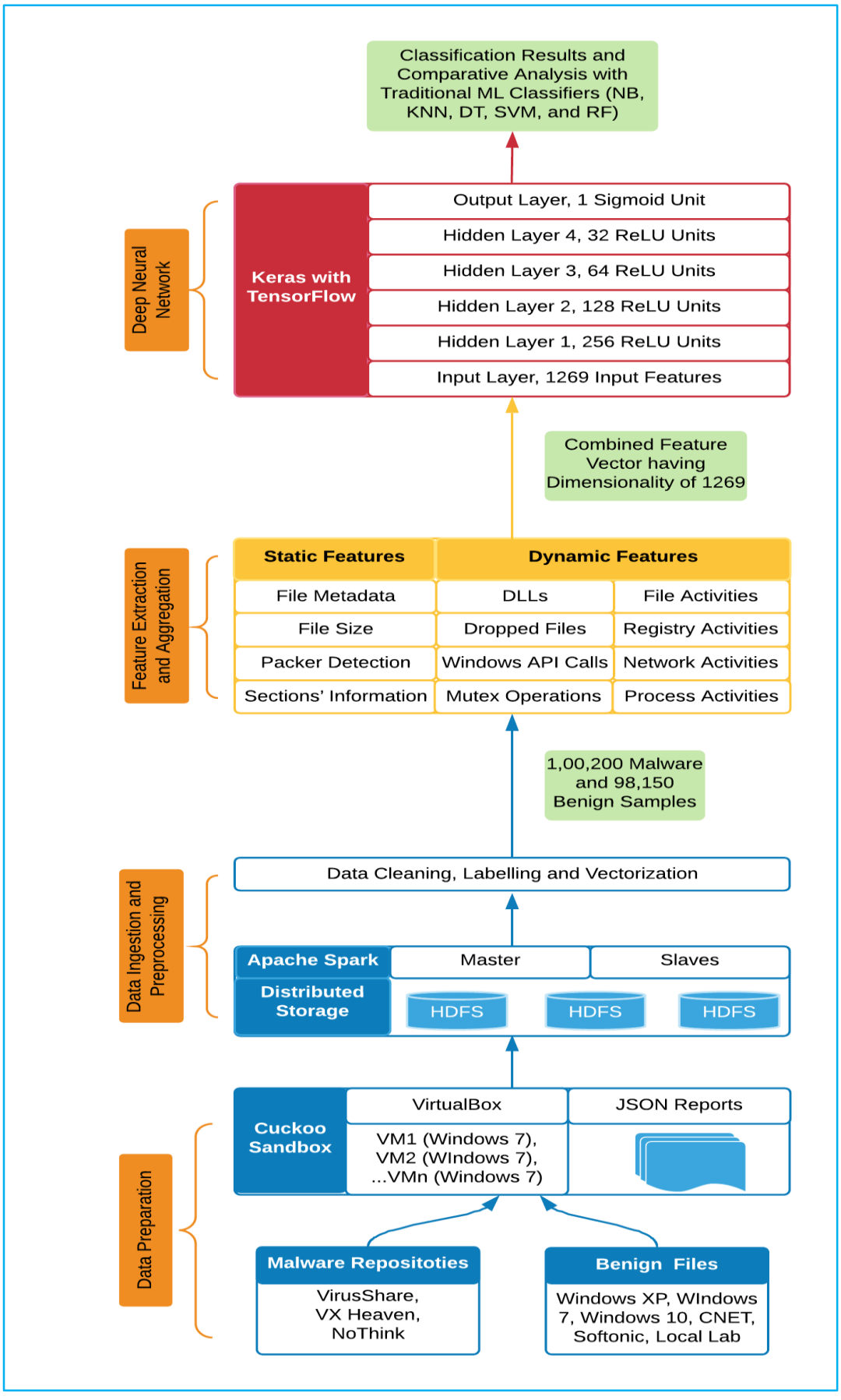


Figure 7.1: High-level overview of proposed model.

NB, KNN, DT, SVM, and RF, and comparison of their performance. Figure 7.1 illustrates the high-level overview of proposed methodology and the various steps are discussed in the following subsections:

7.2.1 Designing Neural Network

DNNs mimic the efficiency and robustness of human brain to process massive data. A neural network model consists of fully connected stacked layers of artificial neurons – an input layer, one or more hidden layers, and an output layer, to learn data representation. The depth of the model generally specifies the number of hidden layers. Each layer of DNN consists of computational nodes called neurons and each neuron is connected to other neuron through edges just like synapses connect real neurons. Each edge has certain weight indicating whether the strength of the connection is excitatory or inhibitory. The behaviour of the network is determined by setting weights for the connections and the learning of a model is indeed the process of finding the appropriate weights through backpropagation [182] to optimize a loss or cost function. Figure 7.2 shows the schematic diagram of proposed DNN.

7.2.2.1 Parameter Initialization

The initialization of the parameters considerably affects the convergence of backpropagation during the training of DNN which is critical for model performance. It is important to select proper values for initialization of weights because too small or large values lead to slow learning and divergence respectively. The rule of thumb is that the mean of activation should be zero and the variance should remain same across the layers [183]. This approach is known as Xavier initialization [184] and is valid for both forward (i.e. activation) as well as backward propagation (i.e. for gradient of cost w.r.t activations). It speeds up the training process and ensures that the backward gradient does not get multiplied by too small or large values and thus, avoids vanishing or exploding gradient problem. The initialization parameters for ReLU activation function are described by equations (7.1) and (7.2).

Normal distribution with zero mean ($\bar{x} = 0$) and standard deviation (SD)

$$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs} + n_{outputs}}} \qquad \ldots (7.1)$$

OR

A uniform distribution (-r, r) with $r = \sqrt{2}\sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$ ... (7.2)

where n_{inputs} and $n_{outputs}$ denote the number of input and output connections of initialized weights respectively.

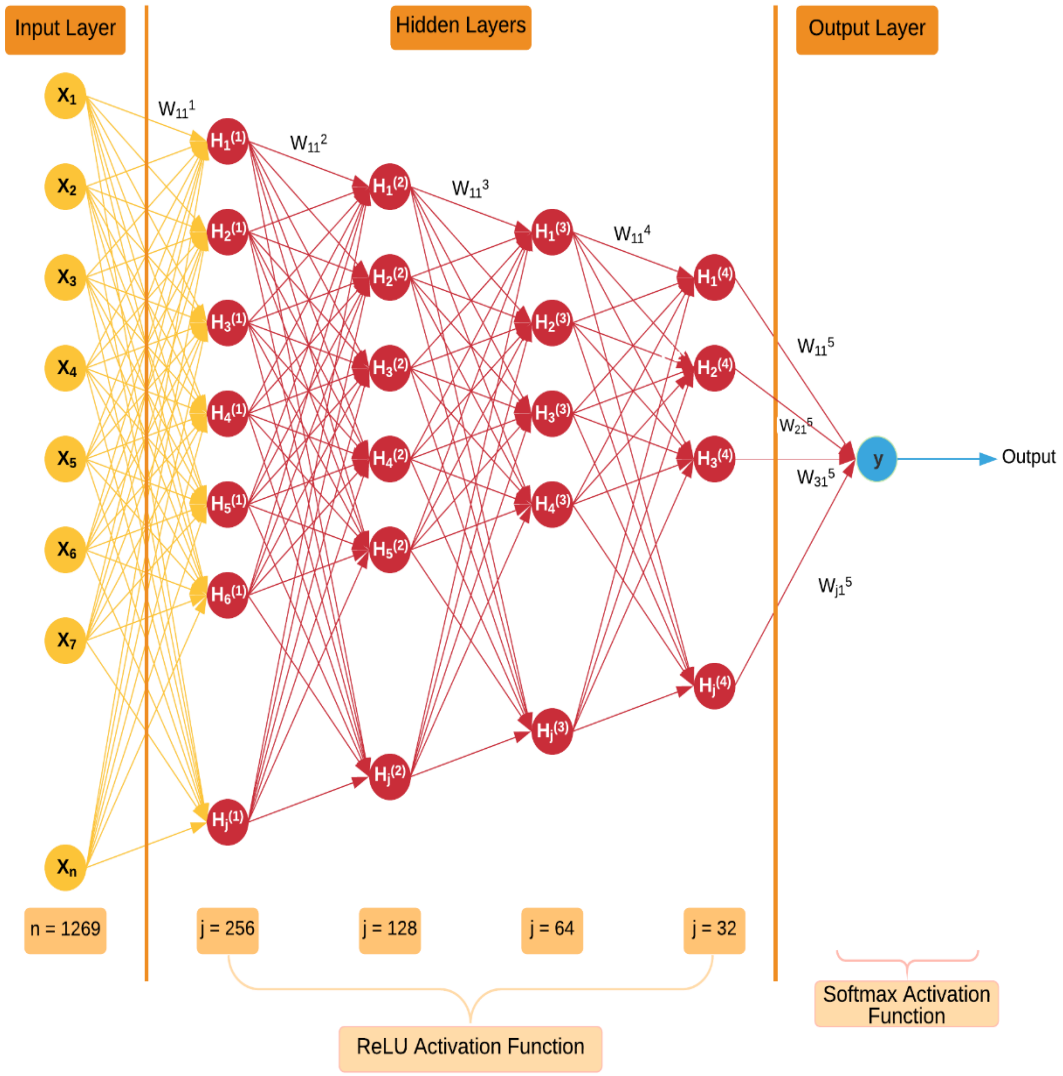


Figure 7.2: Schematic diagram of proposed DNN. The layers are separated by batch normalization and dropout.

7.2.2.2 Selecting Hidden Layers and Neurons

It is difficult to find appropriate number of hidden layers and neurons per hidden layer. In case of hidden layers, one can start with one or two hidden layers and thereafter, gradually increase the layers depending upon the complexity of the problem and model training time, until the model starts experiencing overfitting. Since most of the real-world data is structured in hierarchical way, a model with few layers but large number of neurons takes more time to learn as compared to a model having more layers but lesser number of neurons. In general, it is always advisable to increase number of layers rather than number of neurons per layers.

The number of neurons in the input and output layers depend upon the size of input feature vector and the classification problem. For hidden layers, it is a common practice to choose the number of neurons in such a way that it forms a funnel, i.e., placing lesser and lesser number of neurons in succeeding layers. Another practice is to choose same number of neurons in all hidden layers which requires tweaking of just one hyperparameter. Like hidden layers, it is a good practice to start with fewer neurons and gradually increasing their count until the architecture experiences overfitting. An opposite approach involves large number of layers and neurons than actually required. Thereafter, the model deploys regularization techniques like dropout, early stopping, data augmentation, etc. to avoid overfitting [185].

7.2.2.3 Activation Functions

During training of DNN, a non-linear activation function is deployed in each layer to compute a non-linear transformation of the preceding layer that enables DNN to learn more complex representation of data. The common choices of the activation functions are sigmoid [186], tanh [187], the rectified linear units (ReLU) [188], and exponential linear units (ELU) [189]. We have used ReLU for hidden layers $l = \{1, 2, 3, 4\}$ as it is presently the most widely used activation function. ReLU helps a model to deal with vanishing and exploding gradient problem, and also makes it computationally efficient accelerating the entire training process by reducing number of training epochs. Mathematically, ReLU is defined as shown in equation (7.3).

$$f(x) = \max(0, x) \qquad \ldots (7.3)$$

where x denotes input. The derivative of the ReLU is shown in equation (7.4).

$$ReLU\ (x) = \begin{cases} 0, & if\ x < 0 \\ 1, & if\ x \geq 0 \end{cases} \quad \ldots (7.4)$$

The choice of activation function at the output or last layer $l = 5$ can affect the performance of the model. This layer contains only one neuron and uses sigmoid activation function as shown in equation (7.5) to deliver malware classification result, i.e., it gives either 0 or 1, where 0 denotes benign and 1 malware.

$$f(x) = \frac{1}{1 + e^{-x}} \quad \ldots (7.5)$$

7.2.2.4 Regularization - Handling overfitting

Deep learning models have a natural tendency to be good at fitting the training data well, though the real challenge lies in its generalization to the testing or unseen data which it to make useful predictions. Adding too many layer and nodes per layer or training a model for too long, both result in overfitting that achieves high accuracy on training data, but do not generalize well to the testing data. In contrast, if a model is over-regularized, or simply not trained long enough, .i.e., the network has not learned the required patterns in training data, overfitting may happen. In both cases, the learning model does not generalize well to the testing data. To do so, there is a need devise a method to train the model for an appropriate number of epocs. There are two ways to avoid overfitting - adding more training data, and regularization. The latter helps to generalize data representations by reducing the model size, i.e., the number of learnable parameters which are decided by number of layers and units per layer. In fact, this method restricts the quantity and type of information a model can store so that it can focus only on the most prominent patterns which can generalize well. There is no direct method to decide about model size. It is always better to start with a few layers and parameters, and keep on adding additional layers or increasing their size until validation loss starts showing diminishing returns [77]. It is pertinent to remember that the larger the capacity of the model, the quicker will it train the model but more prone to the overfitting.

To minimize overfitting and improve generalization, DNNs uses various regularization methods like early stopping, weight, dropout [190, 191], gradient noise [192], and

scalable data augmentation [193]. We have used dropout method which randomly leaves pre-defined number of nodes during each epoch while updating the corresponding hidden layer. This method is more resilient in handling of imperfect feature data retrieved from identical malware packed or compiled with diverse applications. We have set a dropout rate of 0.2 which means that 20% of the neurons in each layer are deactivated for each training malware sample. This reduces the dependence of neurons among each other.

7.2.2.5 Loss Function

The loss or cost or objective function describes the closeness of the modeled value to the actual value. In case of high dimensionality, the minimum value of the loss function is achieved by the optimization techniques like gradient (derivative) descent algorithm. The commonly used loss functions are squared error and cross entropy. The latter measures the difference between the probability distributions over same set of classes and is ideally suited for categorical outputs. The cross-entropy loss function is defined using equation (7.6).

$$Loss\left(y_p, y_t\right) = -y_t \ln y_p - (1 - y_t) \ln(1 - y_p) \qquad \ldots (7.6)$$

where $y_p$denotes modeled (or predictor) value, and $y_t \in \{0, 1\}$ denotes the true value, with 0 and 1 denote benign and malware respectively.

The loss function is minimized by repeatedly updating the weights and biases using backpropagation algorithm along with optimization techniques. In proposed model, we deploy both binary cross entropy and Adam optimizer as loss function. Adam [194], a gradient-based optimizer, is computationally efficient method for stochastic optimization. It speeds up the model training as convergence rate is better than standard stochastic gradient descent. The little memory requirement makes it appropriate for large data and parameters. Being an adaptive learning rate algorithm, Adam optimizer involves less tuning of learning rate η and mostly uses its default value 0.001. In addition, the default values for momentum decay β_1, scaling decay β_2, and smoothing term ϵ are set to 0.9, 0.999, and 10^{-8} respectively.

7.3 Evaluation

This section presents the outcomes of proposed model which is implemented using Apache Spark, and Keras along with TensorFlow. An integrated set of static and dynamic features is taken as input to DNN and the outcomes of various hyperparameter configurations are discussed. It is pertinent to remember that the fundamental objective of model learning is to optimize the generalization performance and not training performance [195]. With this intention, we empirically validate the performance generalization by using 10-fold cross validation mechanism and noted the model performance in terms of precision, recall, accuracy, F1-score, and MCC (as discussed in Section 5.5.3).

7.3.1 Dataset

The proposed DNN model for malware detection is evaluated on a dataset of 1,98,350 files including 1,00,200 malicious and 98,150 benign files. Thereafter, an integrated set of static (file metadata, file size, packer detection, and sections' information) and dynamic attributes (DLLs, dropped files, Windows API calls, mutex operations, file activities, registry activities, network activities, and process activities) is considered to create a combined feature vector of dimensionality 1269. This feature vector acts as a raw data to proposed big data framework. The malware data preparation and feature extraction processes have already been described in Sections 4.2.1 and 5.3.1 respectively.

7.3.2 Experimental Environment

The proposed architecture leverage the open-source components including HDFS for distributed storage, Apache Spark for efficient preprocessing of data, and Keras along with TensorFlow as its backend computational engine to develop and train DNN. The advantage of using Apache Spark for data preprocessing is that it happens only once on Spark as compared to TensorFlow graph in which duplicate transformations are done on each training epoch [196]. A brief description of Apache Spark, and other open-source software components is given in Sections 1.4.2 and 4.2.2, while Keras and TensorFlow are discussed under Section 2.3.2. Table 7.1 provides the details of experimental environment.

Table 7.1: Experimental environment.

Category	Description	
Software Tools	Apache Spark	Version: 2.4.0
	Keras	Version: 2.2.4
	TensorFlow	Version: 1.12
	Apache HDFS	Version: 3.1.1
	Python	Version: 3.6.0
	Ubuntu	Version: 18.04.2
Hardware Configuration	Master: Intel(R) Core TM i7-5500U CPU @2.4 GHz, 16 GB DDR3 1600 MHz, and 1 TB HDD	
	Slaves (02): Intel(R) Core TM i5-8250U CPU @1.6 GHz, 8 GB DDR4, and 1 TB HDD	
Dataset	Malware samples: 1,00,200	
	Benign samples: 98,150	
Evaluation Parameters	10-fold cross validation	
	TPR, FPR, FNR, Accuracy, Precision, F-measure, MCC	

7.3.3 Training of Deep Neural Network

The training of a DNN helps a model to learn the parameters and involves the steps like setting initial values of parameters, learning rate, the optimization rules, the regularization, the loss or cost function etc. The main steps of model training are listed below:

1) Initialize the parameters as described in 7.2.2.1
2) Select activation function for hidden as well as output layer as discussed in 7.2.2.3
3) Select an optimization algorithm. We have chosen Adam optimizer as discussed in 7.2.2.5
4) Repeat the steps to minimize the cost function which in turn maximizes the model performance:
   a. Forward propagate an input.
   b. Computer the cost or loss function.
   c. Compute the gradients w.r.t the parameters using backpropagation.
   d. Update the parameters using the gradients based on optimization algorithm as specified in Step 3.

Any iteration loop (i.e. forward propagation, cost calculation, backward propagation, and weights update) as abovementioned in step (4) will either amplify or minimize the backpropagated gradient [183].

7.3.4 Results and Discussion

The performance evaluation provides better understanding of a model in real world. We examine the experimental results using big data and DNN environment for classification of malware dataset. First, DNN model is built using 10-fold cross validation for various hyperparameter configurations. Thereafter, we considered five diverse inducers as base classifiers, namely, NB, KNN, DT, SVM, and RF. These base classifiers are selected from different family of classifiers and use different approach to classify the input data. In Section 6.3.2, RF comes out to be the best base classifier (see Table 6.2). The performance of both DNN and best base classifier model (i.e. RF) is evaluated and compared on the basis of various evaluation parameters like TPR, FPR, FNR, precision, accuracy (%), F-measure, and MCC, which are computed using equations (5.4) to (5.10).

7.3.4.1 Hyperparameter Tuning

At the core, a DNN is a complex mathematical model consisting of millions of hyperparameters which decide the model behavior. The proper selection of hyperparameters makes the model learning process more efficient and ensure optimal usage of computational power. The tuning of hyperparameters is quite challenging, particularly on a large dataset which takes long training time. Different hyperparameter values result in different models and finally, the best performing model is selected. In experiments, the size of the input layer is set to 1269 and based on tuning of hyperparameters, the size of the four hidden layers is set to 256, 128, 64 and 32 respectively. The hidden layers consist of a dropout rate of 0.2 followed by ReLU activation function. Adam optimizer with initial learning rate η of 0.001 is set. The model is trained till the accuracy stops improving further. In experiments, the best hyperparameter configuration (see Table 7.2) includes four hidden layers and the experimental results achieves malware detection rate with 98.7% accuracy using 10-fold cross validation.

Table 7.2: Hyperparameter – possible and best values, for model configuration.

Hyperparameters	Applied Values	Best Value
Hidden layers	2 - 6	4
Neurons per hidden layer	-	256, 128, 64, 32
Batch size	64, 128, 256, 512	128
Epochs	100, 150, 200, 250	150
Dropout rate	0.2	0.2
Learning rate (η)	0.001	0.001

7.3.4.2 Evaluation of DNNs and Comparison with Best Base Classifier

In this subsection, we compare the classification results of RF and DNN with 2, 3, 4, 5, and 6 hidden layers. Table 7.3 illustrates the evaluation results of DNN with four hidden layers report best accuracy of 98.9%, followed by DNN with five and six layers respectively.

Table 7.3: Classification results using base classifiers and DNN.

Classifier	TPR	FPR	FNR	Precision	F-Measure	Accuracy (%)	MCC
DNN (2 hidden layers)	0.973	0.026	0.027	0.975	0.974	97.3	0.947
DNN (3 hidden layers)	0.981	0.02	0.019	0.98	0.98	98	0.961
DNN (4 hidden layers)	0.99	0.013	0.01	0.988	0.989	98.9	0.978
DNN (5 hidden layers)	0.988	0.014	0.012	0.986	0.987	98.7	0.974
DNN (6 hidden layers)	0.987	0.014	0.013	0.986	0.986	98.6	0.973

Figure 7.3 visualizes the FPR/FNR for the base classifiers and DNN. It shows that the values of FPR and FNR are maximum and minimum for DNN with two and four hidden layers respectively. DNN with four hidden layers has delivered highest average accuracy, i.e., 98.9% followed by 98.7% for DNN with five hidden layers (see Figure 7.4).

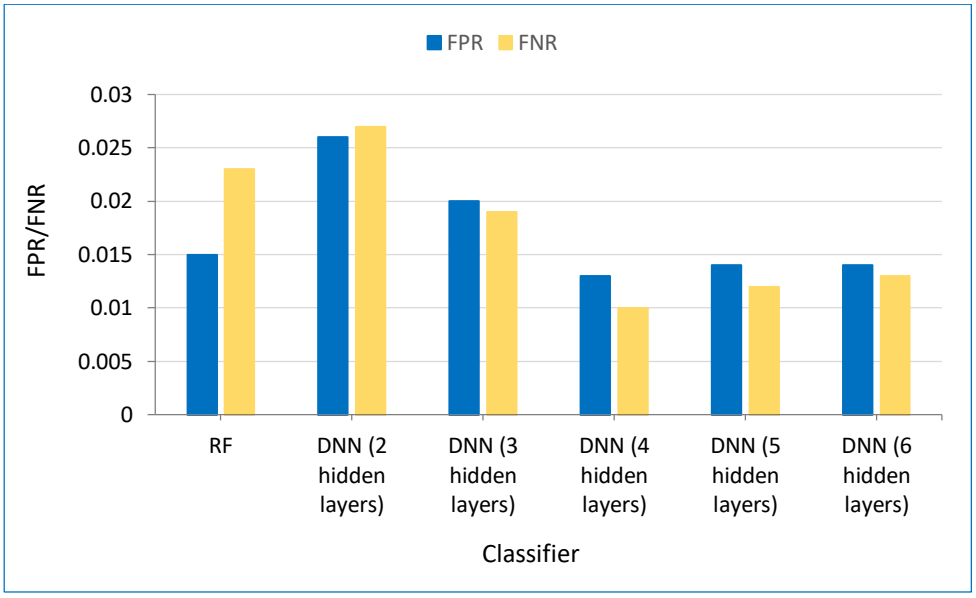


Figure 7.3: Comparison of best base classifier and DNN using FPR/FNR.

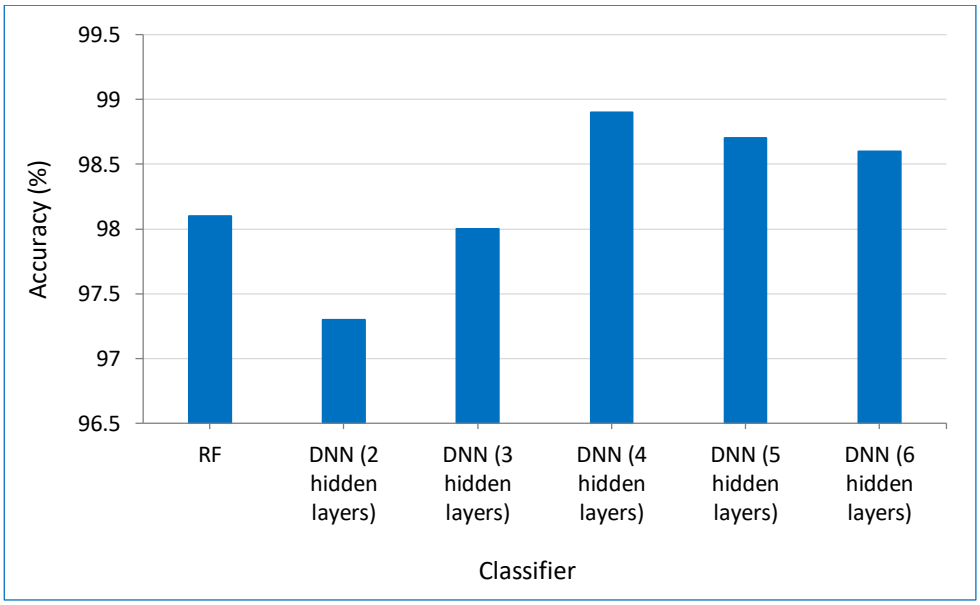


Figure 7.4: Comparison of best base classifier and DNN using accuracy.

Figure 7.5 illustrates that DNN with four hidden layers records the maximum value using MCC, i.e., 0.978 (showing a highest correlation between predicted and the actual data labels) followed by DNN with five hidden layers, i.e., 0.974.

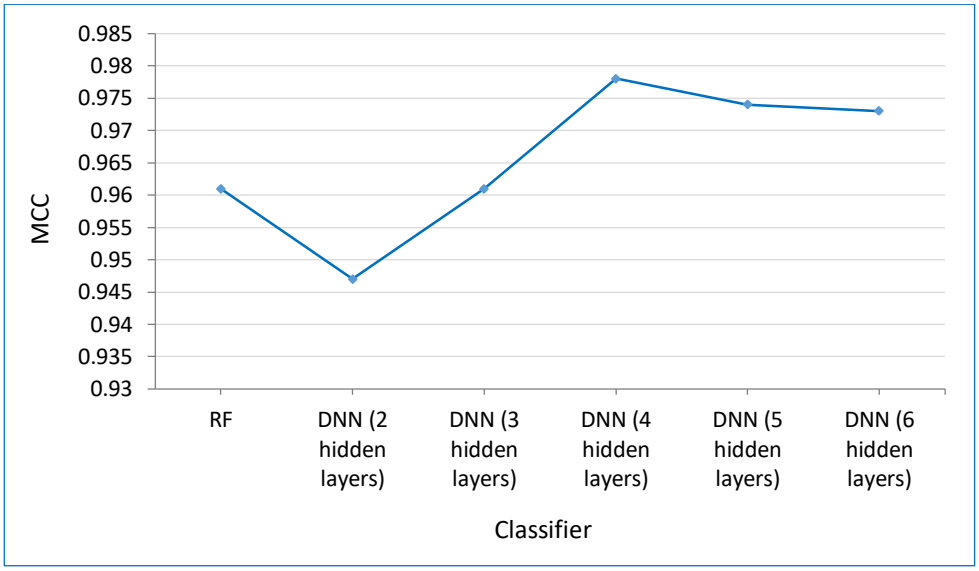


Figure 7.5: Comparison of best base classifier and DNN using MCC.

7.4 Discussion

Malware detection, a major cyber security problem, has seen exponential growth in recent years making it a popular big data use case. The traditional approaches for malware detection have already been proved ineffective to handle massive malware data. Moreover, the traditional machine learning based approaches for malware detection and classification mostly suffer from high FPR and FNR, and thus, the security researchers are turning towards big data and deep learning based solutions.

We have presented a scalable malware classification model based on big data and deep learning technologies. The solution is developed using Apache Spark, and Keras on the top of TensorFlow. The model utilizes an integrated set of static and dynamic features. A total twelve kinds of features are extracted from a dataset of 1,98,350 samples, including 1,00,200 malicious and 98,150 benign files. Further, a 1269 dimensional feature vector space is created and fed into DNN to find the best classification results. In model evaluation, the DNN based model with four hidden layers has achieved a detection rate of 98.9% with 0.013 FPR and 0.01 FNR. These results are compared with best traditional classifier, i.e., RF which confirmed the usefulness of DNN based model.

Deep learning seems to be highly suitable for malware detection problem because the models are robust enough to handle natural variations in malware. Moreover, the scale of deep learning w.r.t the number of training samples, the number of parameters, or both can drastically enhance accuracy of classification models. Consequently, DNN based

malware classification models will gain importance and momentum in the years to come. In future, we aim to build a model based on various deep learning architectures which learns the features (representations) directly from raw data without any feature engineering.

7.5 Summary

This chapter presented the proposed solution for malware detection by bringing together the benefits of Apache Spark – a big data processing engine, and Keras along with TensorFlow, a deep learning framework. Apache Spark is deployed for efficient preprocessing of data, and Keras along with TensorFlow is used to develop and train deep neural network. The resulting model is trained and evaluated on a dataset having integrated set of malware static and dynamic features. The resulting classification model gives better performance than the traditional machine learning algorithms.

Chapter 8: Conclusion and Future Scope

This chapter concludes the thesis by providing a brief overview of the proposed framework and provides insights about the future scope of work. It highlights the contributions made and enlists the points which can be undertaken in future.

8.1 Conclusion

The present research results in the realization of a framework for analyzing big data to generate the actionable insights or intelligence for helping the decision makers. A case study of malware analysis and classification has been used in this work.

After understanding the basic concept of big data, its evolution and popular open-source big data stream processing frameworks, a bibliometric study of academic and industry publications during the period 2000–2017 is conducted to understand the current state, evolving disciplines, tools and techniques, and research trends of big data. A comparison of the most widely used open source big data stream processing frameworks is made, and the major big data research challenges and directions are identified and deliberated to offer rich observations and thinking. With the advent of data science, even more decisions will be made through the knowledge derived from big data and the availability of stream processing frameworks is paving the way to analyze data more quickly than ever. More and more organizations are using Apache Spark for big data implementations and it appears that Apache Spark will finally replace MapReduce as general purpose data processing engine for Apache Hadoop. In general, the big data technologies and techniques are still evolving, and new challenges as well as opportunities are rapidly emerging.

One of the big data problems from cyber security domain is identified as malware classification which has been used as a case study for this research. Firstly, a novel

framework is proposed for analyzing malware data at large scale. Afterwards, it is extended using machine learning and deep learning techniques for providing the actionable insights to the decision makers.

To start with, we have proposed a scalable architecture built on the top of Apache Spark, which is capable of analyzing millions of malicious binaries and providing insights about their behavioral trends. Further, a statistical analysis of 1,70,000 malware reports generated through Cuckoo Sandbox using Windows 7 as the guest machine has been performed to study malware behavioral trends during the period 2010 to 2017. These trends can be further extrapolated by security experts to generate cyber threat intelligence which can help organizations to improve their threat protection systems and reduce the risks posed by malicious binaries. The intelligence so generated can be shared with security experts, CERTs or other stakeholders so that they can issue preventive measures and advisories for future threats to minimize the risks posed by them. This can further facilitate anti-virus industries in selecting important parameters for developing faster and accurate techniques for detecting unknown malicious programs. The analysis of malware behavioral trends obtained using statistical techniques have shown that the different malware families share typical behavioral patterns reflecting their origin and purpose. These patterns can be exploited to detect and classify unknown malwares into their known families using automated machine learning, ensemble learning and deep learning techniques.

Further, we have proposed a scalable architecture built on the top of Apache Spark using its scalable machine learning library MLlib for detecting zero-day malware. The JSON reports generated after performing automated malware analysis are stored in HDFS. Afterwards, the malware features are extracted from these reports and their impact is studied on the classification in order to understand their relative importance. The proposed scalable platform is tested and evaluated for detecting zero-day malware using a dataset covering a large number of malware families over 7 years period (from year 2010 to 2016) of submission to VirusTotal. Three machine learning algorithms, namely, NB, SVM and RF are applied using Apache Spark MLlib library. The experimental results show that RF gives the best accuracy of 98.88% along with the minimum FPR and FNR followed by SVM and NB which gives an accuracy of 94.03% and 89.13% respectively. Thus the proposed framework is able to address the issues and the challenges related to early detection of zero-day malware.

Although, many machine learning models have been used in detection and classification of malicious binaries in literature, however, the performance of ensemble learning methods has not been investigated extensively on large malware data. We have designed two methods based on ensemble learning and big data for improving the performance of malware detection at large scale. The first method uses weighted voting strategy of ensemble learning, and the second method selects an optimal set of base classifiers for stacking purpose. The proposed methods are implemented using Apache Spark, and their performance is tested and evaluated on a dataset having integrated set of static and dynamic features of 1,98,350 Windows files including 1,00,200 malicious and 98,150 benign files. The experiments demonstrate that the proposed approach improves the generalization performance in detecting new malware as compared to traditional ensemble methods.

It has been found that the increasing complexity and sophistication of malware has led to much state-of-art machine learning based solutions. However, many of these solutions suffer from high FPR and low scalability restricting their wider adoption and deployment. In order to deal with this problem, an integrated set of static and dynamic features has been used to develop a deep learning model for unknown malware detection which addresses the issues at large scale. The proposed solution deploys Apache Spark for efficient data preprocessing and Keras with TensorFlow as its computational engine for implementation of deep learning model. The solution achieves malware detection rate at an extremely low FPR and can scale to real world malware volumes on commodity hardware. The findings of 10-cross validation evaluation, based on around 1,98,350 malicious and benign samples, demonstrate that the four layer deep learning model has the highest accuracy of 98.7% detection accuracy with 0.014 FPR and 0.012 FNR.

Therefore, the present research provides an evidence-based knowledge pertaining to application of big data tools in classifying emerging malware using machine learning, ensemble learning and deep learning algorithms. The results obtained after analyzing a large number of malware samples using proposed techniques can be used to bring useful insights about the emerging malware. This can be further deployed to issue advisories, and help anti-virus industries to select appropriate feature set and develop products/solutions to deal with such threats in an effective and efficient manner.

8.2 Research Contributions

The major contributions of the thesis are listed below:

1) A comprehensive study of big data evolution including batch and stream processing tools, bibliometric analysis, and research challenges, is provided.
2) A detailed study of big data security analytics is performed which identified malware detection as a big data problem.
3) Prepared a dataset consisting of 1,70,000 malware binaries by setting up a distributed Cuckoo Sandbox environment with Windows 7 as a guest machine.
4) Identified the issues with existing traditional approaches for malware detection at large scale and proposed the following solutions which are successfully implemented and evaluated in distributed environment:
    - Proposed a big data framework for malware trends analysis to comprehend the changing behavior of malicious binaries in order to extract threat intelligence.
    - A scalable solution for zero-day malware detection on the top of Apache Spark using its scalable library (MLlib) is proposed.
    - Two ensemble learning approaches for malware detection at large scale are proposed. The first approach adopts the weighted voting strategy of ensemble learning, and the second includes selecting an optimal set of base classifiers for stacking purpose. The resulting model improves the performance as compared to traditional ensemble methods.
    - A framework for malware detection at large scale is proposed which brings together the benefits of Apache Spark for efficient data preprocessing, and Kera along with TensorFlow for implementation and training of deep neural network. The framework performs better than the traditional machine learning algorithms.

8.3 Future Scope

The proposed solutions appear promising, however, the future research can incorporate the following tasks:

1) A tremendous amount of data is streaming continuously through the network and thus, the proposed solutions will be extended to handle such data. The ability to process the streaming data at large scale has its own importance in zero-day malware detection. The fast detection and investigation of cyber threats lead to a quick response in near

real-time, which in turn reduces the impact of cyber-attacks and provides better risk management.

2) The efficiency of the proposed detection model in terms of learning time has not been observed in the present study. Besides, the learning time can also be studied and evaluated on the basis of cluster size, dataset size etc.
3) The proposed framework can be offered as a cloud service for malware detection. A hybrid approach may be considered to process malware data on local as well as remote cluster.
4) Different deep learning architectures can be deployed to understand their utility in malware detection problem. Besides, their performance as well as learning time of each architecture shall be evaluated and a comprehensive comparison analysis shall be performed.
5) The adversarial samples can be created by making small perturbations in the original malware samples. Such samples lead to inaccurate detection and thus, these adversely impacts the accuracy of malware detection. The malware detection model should be made intelligent enough to identify and handle such perturbations.
6) Finally, the malware detection model can be extended to cover Android and IoT malware emanating from diverse sources.

References

[1] Company Information. Facebook Newsroom. [Online]. Available: http://newsroom.fb.com/company-info/. Accessed: April 26, 2018.

[2] H. Chaouchi, "The Internet of Things: connecting objects to the web," John Wiley & Sons, 2013.

[3] T. Hey, S. Tansley and K. Tolle, Eds., "The Fourth Paradigm - Data Intensive Scientific Discovery," Microsoft Corporation, 2009. [Online]. Available: https://www.microsoft.com/en-us/research/publication/fourth-paradigm-data-intensive-scientific-discovery/. Accessed: March 25, 2017.

[4] Z.D. Stephens, S.Y. Lee, F. Faghri, R.H. Campbell, C. Zhai, M.J. Efron, R. Iyer, M.C. Schatz, S. Sinha, and G.E. Robinson, "Big data: astronomical or genomical?," *PLoS biology*, vol. 13, no. 7, p.e1002195, 2015.

[5] CERN. The Large Hadron Collider, [Online]. Available: http://home.cern/topics/large-hadron-collider. Accessed: March 23, 2017.

[6] H.R., Varian, "Big Data: New Tricks for Econometrics," *Journal of Economic Perspectives*, vol. 28, pp. 3-27, 2014.

[7] Big Data, Nature, 2008. [Online]. Available: http://www.nature.com/news/specials/bigdata/index.html. Accessed: February 12, 2017.

[8] Special Online Collection: Dealing with Data *Science*, 2011. [Online]. Available: http://www.sciencemag.org/site/special/data/. Accessed: February 12, 2017.

[9] S. Lohr, "The age of big data," *The New York Times*, 2012. [Online]. Available: http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?pagewanted=all. Accessed: February 16, 2017.

[10] Data, data everywhere. *The Economist, 2010*. [Online]. Available: http://www.economist.com/node/15557443. Accessed: February 26, 2017.

[11] The Networking and Information Technology Research and Development Program (NITRDP). Big Data Interagency Working Group (IWG). [Online]. Available: https://www.nitrd.gov/nitrdgroups/index.php?title=Big_Data. Accessed: January 12, 2017.

[12] Department of Science and Technology. Big Data Initiative. [Online]. Available: http://dst.gov.in/big-data-initiative-1. Accessed: January 12, 2017.

[13] United Nations Global Pulse. Harnessing big data for development and humanitarian action. [Online]. Available: http://unglobalpulse.org/. Accessed: January 10, 2017.

[14] F.X. Diebold, "A Personal Perspective on the Origin(s) and Development of "Big Data": The Phenomenon, the Term, and the Discipline," *Pier Working Paper Archive,* Second Version. Penn Institute for Economic Research, Department of Economics, University of Pennsylvania, 2012.

[15] D. Laney D, "3D data management: Controlling data volume, velocity and variety," *META Group Research Note*," vol. 6, no. 70, p.1., 2001.

[16] W.L. Chang, "NIST Big Data Interoperability Framework: Volume 1, Definitions. Special Publication (NIST SP)-1500-1," 2015.

[17] Berkeley School of Information, "What Is Big Data?," [Online]. Available: https://datascience.berkeley.edu/what-is-big-data/. Accessed: February 21, 2017.

[18] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A.H. Byers, "Big data: The next frontier for innovation, competition, and productivity, McKinsey Global Institute Report. 2011

[19] Gartner IT Glossary, "Big Data (definition)," [Online]. Available: http://www.gartner.com/it-glossary/big-data. Accessed: February 25, 2017.

[20] V. Mayer-Schönberger, K. Cukier, "Big data: A revolution that will transform how we live, work, and think," Houghton Mifflin Harcourt, 2013.

[21] D. Vesset, C. Gopal, D. Schubmehl, C. W. Olofson, S. Bond, M. Fleming, C. Solinger, H. Schwenk, P. Carnelley, and C. K. Chua. IDC's Worldwide Big Data and Analytics Software Taxonomy. *IDC*. March 2017.

[22] D. Vesset, D. Schubmehl, M. Fleming, P. Carnelley, H. Schwenk, M. Webster, C. Kan Chua, T. Manabe, M. Perry, J. Pina, and M. Kumar, IDC FutureScape: Worldwide Analytics, Cognitive/AI, and Big Data 2017 Predictions. *IDC*. 2016.

[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction

for in-memory cluster computing," *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2012, pp. 2-2.

[24] Apache Hadoop, HDFS Architecture, [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Accessed: January 13, 2017.

[25] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107-113, 2008.

[26] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, and B. Saha, "Apache hadoop yarn: Yet another resource negotiator," *Proceedings of the 4th annual Symposium on Cloud Computing*, ACM, October 2013, p. 5.

[27] S. Ghemawat, H. Gobioff, S.T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29-43, 2003.

[28] K. Shim, "MapReduce algorithms for big data analysis," In: *VLDB Endowment*, vol. 5, pp. 2016-2017, 2012.

[29] K.H. Lee, Y.J. Lee, H. Choi, Y.D. Chung, and B. Moon, "Parallel data processing with MapReduce: a survey," *ACM SIGMoD Record*, vol. 40, pp. 11-20, 2012.

[30] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel DBMSs: friends or foes?," *Communications of the ACM*, vol. *53*, *no.* 1, pp.64-71. 2010.

[31] J. Dean, S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM*. vol. 53, pp. 72-77, 2010.

[32] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud, vol.* 10 (10–10), pp. 95, 2010.

[33] Sort Benchmark. [Online]. Availabe: http://sortbenchmark.org/. Accessed: February 10, 2017.

[34] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J.M. Patel, S. Kulkarni, J. Jackson, K. Gade, K., M. Fu, J. Donham, and N. Bhagat, "Storm@ twitter," *Proceedings of the 2014 ACM SIGMOD international conference on Management of data, ACM,* 18 June 2014, pp. 147-156.

[35] Apache Samza. [Online]. Available: http://samza.apache.org/. Accessed: June 12, 2017.

[36] P. Carbone, S. Ewen, and S. Haridi, “Apache Flink™: Stream and Batch Processing in a Single Engine,” *IEEE Data Engineering Bull*, vol. 38, pp. 28-38, 2015.

[37] Apache Apex. [Online]. Available: https://apex.apache.org/. Accessed: June 12, 2017.

[38] Benchmarking Streaming Computation Engines at Yahoo. [Online]. Available: https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at. Accessed: February 26, 2017.

[39] Data Artisans, Extending the Yahoo! Streaming Benchmark. [Online]. Available: https://data-artisans.com/extending-the-yahoo-streaming-benchmark/. Accessed: March 15, 2017.

[40] Data Torrent, “Implementing Linear Road Benchmark in Apache Apex,” [Online]. Available: https://www.datatorrent.com/blog/blog-implementing-linear-road-benchmark-in-apex/. Accessed: February 8, 2017.

[41] Y. Wang, “Stream Processing Systems Benchmark: StreamBench,” Master’s Thesis, Aalto University, 2016.

[42] Google Trends. [Online]. Available: https://trends.google.com/trends/. Accessed: April 12, 2018.

[43] T.C. Redman, “Bad Data Costs the U.S. $3 Trillion Per Year,” *Harvard Business Review*, pp. 11-18, 2016.

[44] B. Saha and D. Srivastava, “Data quality: The other Face of Big Data,” *In 30th International Conference on Data Engineering (ICDE)*, 2014 IEEE. 31 March – 4 April 2014, pp. 1294-1297.

[45] S. Sarsfield, “The Butterfly Effect of Data Quality,” *In the 5th MIT Information Quality Industry Symposium*, 2011.

[46] The White House, Executive Order -- Creating a National Strategic Computing Initiative. The White House. 29 July, 2015. [Online]. Available: https://obamawhitehouse.archives.gov/the-press-office/2015/07/29/executive-order-creating-national-strategic-computing-initiative. Accessed: February 8, 2017.

[47] CISCO, “The Zettabyte Era: Trends and Analysis Cisco,” 02 June, 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Accessed: February 26, 2017.

[48] C. Rudin, D. Dunson, R. Irizarry R, H. Ji, E. Laber, J. Leek, T. McCormick, S. Rose, C.Schafer, M. van der Laan, L. Wasserman, L. Xue, “Discovery with Data -

Leveraging Statistics with Computer Science to Transform Science and Society," *American Statistical Association*, July, 2014.

[49] DB-Engines Ranking. [Online]. Available: http://db-engines.com/en/ranking. Accessed: February 23, 2017.

[50] N. Yuhanna, G. Leganza, and C. Austin, "The Forrester Wave™: Big Data NoSQL," Q3 2016.

[51] F. Matsebula, and E. Mnkandla, "A big data architecture for learning analytics in higher education," In *2017 IEEE AFRICON*, IEEE, 2017, pp. 951-956.

[52] M. Hilbert, "Big Data for Development: From Information – to Knowledge Societies," 2013.

[53] S. Chaudhuri, U. Dayal, V. Narasayya, "An overview of business intelligence technology," *Communications of the ACM*, vol. 54, no. 8, pp. 88-98, 2011.

[54] S. Chaudhuri, "What next?: a half-dozen data management research goals for big data and the cloud," *In 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pp. 1-4, 2012.

[55] Gantz J, D. Reinsel, "The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East," IDC iView: IDC Analyze the Future. 2012.

[56] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou, "Big data challenge: a data management perspective," *Frontiers of Computer Science*, vol. 7, no. 2, pp. 157-164, 2013.

[57] X. Wang, and Y. He, "Learning from Uncertainty for Big Data: Future Analytical Challenges and Strategies," *IEEE Systems, Man, and Cybernetics Magazine, vol. 2, no. 2, pp. 26-31*, 2016.

[58] N. Kashyap, and V. Bhattacharjee, "Security in Privacy Preserving Data Mining," *International Journal of Engineering And Computer Science*, Vol. 4, no. 4, pp. 11698-11703, 2015.

[59] D. Agrawal, P. Bernstein, E. Bertino, "Challenges and Opportunities with Big Data - A community white paper developed by leading researchers across the United States," 2012. [Online]. Available: http://cra.org/ccc/docs/init/bigdatawhitepaper .pdf. Accessed: March 26, 2017.

[60] NITRD. The Federal Big Data Research and Development Strategic Plan. May, 2016.

[61] G. Kapil, A. Agrawal and R.A. Khan, “Security Challenges and Precautionary Measures: Big Data Perspective,” *ICIC Express Letters*, vol. 12, no. 9, pp. 947-954, 2018.

[62] R. Agrawal, A. Kadadi, X. Dai, and F. Andres, “Challenges and Opportunities with Big Data Visualization,” *Proceedings of the 7th International Conference on Management of computational and collective intelligence in Digital EcoSystems*, ACM, 25 October 2015. pp. 169-173.

[63] Symantec, Internet Security Threat Report 2017, Volume 22. [Online]. Available: https://digitalhubshare.symantec.com/content/dam/Atlantis/campaigns-and-launches/FY17/Threat%20Protection/ISTR22_Main-FINAL-JUN8.pdf?aid=elq. Accessed: March 30, 2018.

[64] Intel, A Guide to the Internet of Things Infographic, 2017. [Online]. Available: https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html. Accessed: March 30, 2018.

[65] D. Reinsel, J. Gantz, J. Rydning, “Data Age 2025: The Evolution of Data to Life-Critical,” An IDC White Paper, 2017.

[66] Cisco, “Annual Cybersecurity Report,” 2018.

[67] Gartner, The Fast-Evolving State of Security Analytics, 2016. [Online]. Available: https://www.gartner.com/doc/3274217/fastevolving-state-security-analytics. Accessed: February 22, 2018.

[68] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, p. 436, 2015.

[69] K. Cho, A. Courville, and Y. Bengio, “Describing multimedia content using attention-based encoder-decoder networks." *IEEE Transactions on Multimedia, vol.* 17, no. 11, pp. 1875-1886, 2015.

[70] Y. T. Zhuang, F. Wu, C. Chen, and Y.H. Pan, “Challenges and opportunities: from big data to knowledge in AI 2.0.,” *Frontiers of Information Technology & Electronic Engineering*, vol. *18 no.*1, pp.3-14, 2017.

[71] A. Sina, A. Aeenmehr, and H. Mohamadian, “Power system voltage control using lp and artificial neural network,” *World Acad Sci Eng Technol*, vol. *6*, pp.560-564, 2012.

[72] I. Goodfellow, Y. Bengio, A. Courville, “Deep learning,” MIT press, 2016.

[73] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia,

and A. Talwalkar, “Mllib: Machine learning in apache spark,” *Journal of Machine Learning Research, vol.* 17, no. 34, pp. 1–7, 2016.

[74] G. D. F. Morales, and A. Bifet, “SAMOA: scalable advanced massive online analysis,” *Journal of Machine Learning Research*,” vol. 16, no. 1, pp. 149-153, 2015.

[75] D. Sculley, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, “Machine learning: The high-interest credit card of technical debt,” 2014.

[76] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, “Tensorflow: A system for large-scale machine learning,” *In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265-283.

[77] C. Francois, “Deep learning with Python,” Manning, 2017.

[78] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, “Automatic differentiation in PyTorch,”, 2017.

[79] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” *arXiv preprint arXiv:1512.01274*, 2015.

[80] A. Sergeev, and M. Del Balso, “Horovod: fast and easy distributed deep learning in TensorFlow,” *arXiv preprint arXiv:1802.05799*, 2018.

[81] D. Yu, K. Yao, Y. Zhang, “The computational network toolkit,” *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp.123-126, 2015.

[82] Benchmarking State-of-the-Art Deep Learning Software Tools. [Online]. Available: http://dlbench.comp.hkbu.edu.hk/ Accessed: March 18, 2019.

[83] S. Tokui, K. Oono, S. Hido, J. Clayton, “Chainer: a next-generation open source framework for deep learning,” *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, 2015 Dec 7, pp. 1-6.

[84] A. Sathi, "Big Data Analytics: Disruptive Technologies for Changing the Game,” *Boise: Mc Press*, 2012.

[85] T. Mahmood, and U. Afzal, “Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools,” *In 2nd International Conference on Information assurance*, IEEE, 2013, pp. 129-134.

[86] E. Chickowski, “A case study in security big data analysis,” *Dark Reading*, 2012.

[87] P. K. Manadhata, S. Yadav, P. Rao, and W. Horne, “Detecting malicious domains via graph inference," In *European Symposium on Research in Computer Security*, Springer, Cham, 2014, pp. 1-18.

[88] M.C. Marco, and B. Yolant, “Big Data for Security: Usable DNS Data Analytics at Scale,” *Hewlett Packard Labs*, 2016.

[89] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, “Botcloud: Detecting botnets using mapreduce,” *In IEEE International Workshop on Information Forensics and Security (WIFS), IEEE,* 2011, pp. 1-6.

[90] J. François, S. Wang, and T. Engel, “BotTrack: tracking botnets using NetFlow and PageRank. *In International Conference on Research in Networking*, Springer, Berlin, Heidelberg, 2011, pp. 1-14.

[91] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, “Big data analytics framework for peer-to-peer botnet detection using random forests,” *Information Sciences,* vol. 278, pp. 488–497, 2014.

[92] A.A. Cardenas, P.K. Manadhata, and S.P. Rajan, “Big data analytics for security,” *IEEE Security & Privacy,* vol. 11, no. 6, pp. 74-76, 2013.

[93] P. Giura, and W. Wang, “Using large scale distributed computing to unveil advanced persistent threats,” *Science Journal,* vol. 1, no.3, pp. 93-105, 2012.

[94] T.F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, “Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks,” *Proceedings of the 29th Annual Computer Security Applications Conference*, ACM, pp. 199-208, 2013.

[95] J. Blankenship, S. Balaouras, B. Barringham, and P. Harrison. 2017. The Forrester Wave™: Security Analytics Platforms, Q1 2017. *Forrester*.

[96] IBM, QRadar Security Intelligence, [Online]. Available: https://ibm.com/security/security-intelligence/qradar. Accessed: January 16, 2018.

[97] Splunk, Splunk Enterprise Security. Analytics-Driven SIEM Security, [Online]. Available: https://splunk.com/en_us/products/premium-solutions/splunk-enterprise-security.html. Accessed: July 16, 2017.

[98] Splunk, “Machine Learning, The next big thing in big data,” 2017. [Online]. Available: https://splunk.com/en_us/resources/machine-learning.html. Accessed: July 16, 2017.

[99] LogRhythm, The Security Intelligence Company. [Online]. Available: https://logrhythm.com/products/security-intelligence-platform. Accessed: July 16, 2017.

[100] RSA, Security & Behavioral Analytics. [Online]. Available: https://www.rsa.com/en-us/products/threat-detection-and-response/security-and-behavioral-analytics. Accessed: July 16, 2017.

[101] SNYPR, SNYPR Security Analytics Platform. [Online]. Available: https://securonix.com/snypr-security-analytics-platform. Accessed: July 16, 2017.

[102] McAfee, McAfee Labs Threats Report, 2015. [Online]. Available: http://www.mcafee.com/in/security-awareness/articles/mcafee-labs-threats-report-aug-2015.aspx. Accessed: July 16, 2017.

[103] Quick heal, Quick heal annual threat report, 2019. [Online]. Available: https://www.quickheal.co.in/documents/threat-report/QH-Annual-Threat-Report-2019.pdf. Accessed: May 16, 2019.

[104] CERT-EU, "WannaCry Ransomware Campaign Exploiting SMB Vulnerability" 2017. [Online]. Available: https://cert.europa.eu/static/SecurityAdvisories/2017/CERT-EU-SA2017-012.pdf. Accessed: March 20, 2018.

[105] Addressing Big Data Security Challenges: The right tools for smart protection, 2012. [Online]. Available: http://www.trendmicro.co.in/cloud-content/us/pdfs/about/wp_big-data-security-challenges.pdf. Accessed: April 16, 2014.

[106] U. Bayer, A. Moser, C. Krugel, and E. Kirda, "Dynamic analysis of malicious code," *Journal of Computer Virology*, vol. 2, no. 1, pp. 67-77, 2006.

[107] P. Mell, K. Kent, and J. Nusbaum, "Guide to Malware Incident Prevention and Handling," *National Institute of Standards and Technology (NIST)*, 2005.

[108] E. Skoudis, and L. Zeltser, "Malware: Fighting Malicious Code," *Upper Saddle River, NJ, Prentice Hall PTR*, 2004.

[109] M. Sikorski, and A. Honig. 2012. *Practical malware analysis: The hands-on guide to dissecting malicious software*. No Starch Press.

[110] U. Bayer, E. Kirda, and C. Kruegel, "Improving the efficiency of dynamic malware analysis," *Proceedings of the 2010 ACM Symposium on Applied Computing*, ACM, 2010, pp. 1871-1878.

[111] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," *Proceedings. of* 2001 *IEEE Symposium on Security and Privacy*, Oakland, May, 2001, pp. 38-49.

[112] J. Z. Kolter, and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," *Proceedings. of the* 10*th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 2004, pp. 470-478

[113] M. Siddiqui, M.C. Wang, and J. Lee, "Detecting internet worms using data mining techniques," *Journal of Systemics, Cybernetics and Informatics*, vol. 6, no. 6, pp.48-53. 2009.

[114] R. Tian, L. Batten, and S. Versteeg, "Function Length as a Tool for Malware Classification," *Proceedings of the* 3*rd International Conference on Malicious and Unwanted Software*, Fairfax, pp. 57-64, 2008.

[115] I. Santos, L. Carlos, and P.G. Bringas. "Collective classification for unknown malware detection," *Proceedings of the International Conference on Security and Cryptography (SECRYPT),* IEEE, 2011, pp. 251-256.

[116] A. Saini, E. Gandotra, D. Bansal, and S. Sofat, "Classification of PE files using static analysis," *Proceedings of the 7th International Conference on Security of Information and Networks, ACM,* 2014, pp. 429.

[117] X. Hu, J. Jang, T. Wang, Z. Ashraf, M. Ph Stoecklin, and D. Kirat. "Scalable malware classification with multifaceted content features and threat intelligence," *IBM Journal of Research and Development, vol.* 60, no. 4, pp. 6-10, 2016.

[118] Moser, Andreas, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection." In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pp. 421-430. IEEE, 2007.

[119] R. Tian, M.R. Islam, L. Batten, and S. Versteeg, "Differentiating Malware from Cleanwares Using Behavioral Analysis," *Proc. of* 5*th International Conference on Malicious and Unwanted Software* (*Malware*), IEEE, Nancy, October 2010, pp. 23-30.

[120] S. Nari, and A. A. Ghorbani, "Automated malware classification based on network behavior," In *2013 International Conference on Computing, Networking and Communications (ICNC)*, IEEE, 2013, pp. 642-647.

[121] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, ACM, 2010, pp. 45.

[122] M. Imran, M., M.T. Afzal, and M.A. Qadir, "Using hidden markov model for dynamic malware analysis: First impressions," *Proceedings of 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, IEEE, 2015, pp. 816-821.

[123] B. Anderson, S. Curtis and L. Terran, "Improving malware classification: bridging the static/dynamic gap," *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, ACM, 2012, pp. 3-14.

[124] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P.G Bringas, "Opem: A static dynamic approach for machine-learning-based malware detection," *in International Joint Conference CISIS12-ICEUTE_ 12-SOCO_ 12 Special Sessions. Springer*, 2013, pp. 271-280.

[125] R. Islam, R. Tian, L.M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of network and computer applications*, vol. 36, no. 2, pp. 646-656, 2013.

[126] E. Gandotra, D. Bansal, and S. Sofat, "Integrated framework for classification of malwares," i*n Proceedings of the 7th International Conference on Security of Information and Networks.* ACM, 2014, pp. 417.

[127] A. Damodaran, F. Di Troia, C.A. Visaggio, T.H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp.1-12, 2017.

[128] S.H. Ahn, N.U. Kim, and T.M. Chung, "Big data analysis system concept for detecting unknown attacks," In *16th International Conference on Advanced Communication Technology*, IEEE, 2014, pp. 269-272.

[129] K.Rai, M.S. Devi and A. Guleria, "Decision Tree Based Algorithm for Intrusion Detection," *Int. J. Advanced Networking and Applications*, vol. 7, no. 4, pp. 2828-2834, 2016.

[130] M. Chong, A. Abraham, and M. Paprzycki, "Traffic accident data mining using machine learning paradigms," In *Fourth International Conference on Intelligent Systems Design and Applications (ISDA'04), Hungary,* August 2004, pp. 415-420.

[131] C.R. Panigrahi, M. Tiwari, B. Pati, and R. Prasath, "Malware detection in big data using fast pattern matching: A hadoop based comparison on GPU," In *Mining Intelligence and Knowledge Exploration*, Springer, Cham, pp. 407-416, 2014.

[132] B.A. Miller, "*Scalable platform for malicious content detection integrating machine learning and manual review,"* Doctoral dissertation, UC Berkeley, 2015.

[133] B. Zhang, J. Yin, J. Hao, J., D. Zhang, and S. Wang, "Malicious codes detection based on ensemble learning," *In International Conference on Autonomic and Trusted Computing,* Springer, Berlin, Heidelberg, July, 2007, pp. 468-477.

[134] S. Mukkamala, A.H. Sung, and A. Abraham, "Intrusion detection using an ensemble of intelligent paradigms," *Journal of network and computer applications,* vol. *28, issue* 2, pp.167-182. 2005.

[135] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis,* vol. 53, no. 4, pp. 1483–1494, 2009.

[136] Y. Ye, T. Li, Q. Jiang, Z. Han, L. Wan, "Intelligent file scoring system for malware detection from the gray list," *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, June, 2009, pp. 1385-1394.

[137] S. Guo, Q. Yuan, F. Lin, F. Wang, and T. Ban, "A malware detection algorithm based on multi-view fusion," *In International Conference on Neural Information Processing,* Springer, Berlin, Heidelberg, Nov. 2010, pp. 259-266.

[138] J. Landage, J. and M.P. Wankhade, "Malware detection with different voting schemes," *Compusoft,* vol. 3, no. 1, pp. 450-456, 2014.

[139] S. Sheen, R. Anitha, and P. Sirisha, "Malware detection by pruning of parallel ensembles using harmony search," Pattern Recognition Letters, vol. 34, no. 14, pp. 1679-1686, 2013.

[140] B. Krawczyk, and M. Woźniak, "Evolutionary cost-sensitive ensemble for malware detection," *In International Joint Conference SOCO'14-CISIS'14-ICEUTE'14,* Springer, Cham, 2014, pp. 433-442.

[141] M. Ozdemir, I. Sogukpinar, "An android malware detection architecture based on ensemble learning," *Transactions on Machine Learning and Artificial Intelligence,* vol. 2, no. 3, pp. 90-106, 2014.

[142] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing,* vol. 151, pp. 905-912, 2015.

[143] S.Y. Yerima, and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," *IEEE transactions on cybernetics*, pp. 1-14, 2018.

[144] J. Bai, and J. Wang, "Improving malware detection using multi-view ensemble learning," *Security and Communication Networks*, vol. 9, no. 17, pp. 4227-4241, 2016.

[145] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," *In Proceedings of the twenty-first international conference on Machine learning*, ACM, July, 2004, p. 18.

[146] K. Li, "Diversity in multiple classifier systems (Editorial)," *Information Fusion*, vol. 6, no. 1, 2005.

[147] J.O. Kephart, "A biologically inspired immune system for computers," *In Artificial Life IV: proceedings of the fourth international workshop on the synthesis and simulation of living systems*, July, 1994, pp. 130-139.

[148] G.E. Dahl, J.W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," *In International Conference on Acoustics, Speech and Signal Processing*, IEEE, May, 2013, pp. 3422-3426.

[149] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," In *2015 International Joint Conference on. Neural Networks (IJCNN), IEEE*, 2015, pp. 1–8.

[150] R. Pascanu, J.W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, April, 2015, pp. 1916-1920.

[151] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," In *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), IEEE, October, 2016*, pp. 104-111.

[152] J. Saxe, and B. Konstantin, "Deep neural network based malware detection using two dimensional binary program features," In *10th International Conference on Malicious and Unwanted Software (MALWARE)*, IEEE, 2015, pp. 11-20.

[153] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," In *Australasian Joint Conference on Artificial Intelligence*, Springer, 2016, pp. 137–149.

[154] W. Huang, J.W. Stokes, "MtNet: a multi-task neural network for dynamic malware classification," *In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, Cham*, July 2016, pp. 399-418.

[155] R. Sun, X. Yuan, P. He, Q. Zhu, A. Chen, A. Gregio, D. Oliveira, and X. Li, "Learning fast and slow: Propedeutica for real-time malware detection," *arXiv preprint arXiv:1712.01145*, 2017.

[156] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "DeepAM: a heterogeneous deep learning framework for intelligent malware detection. *Knowledge and Information Systems*, vol. *54, no.* 2, pp.265-285, 2018.

[157] C. Wu, J. Shi, Y. Yang, W. Li, "Enhancing Machine Learning Based Malware Detection Model by Reinforcement Learning," *Proceedings of the 8th International Conference on Communication and Network Security, ACM, November 2,* 2018, pp. 74-78.

[158] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Computers & Security*, vol. 77, pp. 578-594, 2018.

[159] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats. Boston, 2009.*

[160] E. Gandotra, D. Bansal, and S. Sofat, "Malware intelligence: beyond malware analysis," *International Journal of Advanced Intelligence Paradigms*, vol. *13,* pp.80-100, 2019.

[161] I. Daisuke, K. Yoshioka, M. Eto, Y. Hoshizawa, and K. Nakao, "Malware behavior analysis in isolated miniature network for revealing malware's network activity," *In International Conference on Communications, IEEE,* 2008, pp. 1715-1721.

[162] C. Lorenzo, C. Kruegel, G. Vigna, F. Yu, M. Alkhalaf, T. Bultan, L. Cao, L. yang, H. Zheng, C.C. Cipriano, and T.F. Gonzalez, "Mining the network behavior of bots. *Technical Report*, 2009, pp. 2009-12.

[163] R. Christian, C.J. Dietrich, H. Bos, L. Cavallaro, M.V. Steen, F.C. Freiling, and N. Pohlmann, "Sandnet: Network traffic analysis of malicious software," *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ACM, 2011, pp. 78-88.

[164] Q. J. Snaker, PE iDentifier (PEiD), 2008. [Online]. Available: https://tuts4you.com/download.php?view.398. Accessed: June 23, 2017.

[165] J. Han, J. Pei and M. Kamber, “Data Mining: Concepts and Techniques, Elsevier, 2011.

[166] C. Cortes, and V. Vapnik, “Support-Vector Networks,” *Machine learning,* vol 20, no. 3, pp. 273-297, 1995.

[167] L. Breiman, “Random forests,” *Machine learning,* vol 45, no. 1, pp. 5-32, 2001.

[168] S. Raschka, “About feature scaling and normalization, Disques, nd Web, 2014.

[169] C.E. Shannon, “A mathematical theory of communication,” *Bell system technical journal*, vol. 27, no. 3, pp. 379-423, 1948.

[170] Y. Ye, D. Wang, T. Li, D. and D. Ye, “IMDS: Intelligent malware detection system,” *In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2007, pp. 1043-1047.

[171] D.V. Sang, D. M. Cuong, and L.T.B Cuong, “An Effective Ensemble Deep Learning Framework for Malware Detection,” *Proceedings of the Ninth International Symposium on Information and Communication Technology*, ACM. 2018, pp. 192-199.

[172] B.W. Matthews, “Comparison of the predicted and observed secondary structure of T4 phage lysozyme,” *Biochimica et Biophysica Acta (BBA)-Protein Structure,* vol. 405, no. 2, pp. 442-451, 1975.

[173] G. James, D. Witten, T. Hastie, and R. Tibshirani, “An introduction to statistical learning,” *New York: springer,* vol. 112, no. 18. 2013.

[174] M.A. Maloof ed., “Machine learning and data mining for computer security: methods and applications,” *Springer Science & Business Media,* 2006.

[175] L. I. Kuncheva, “Combining pattern classifiers: methods and algorithms,” *John Wiley & Sons,* 2004.

[176] D. H. Woplert, “Stacked generalization,” *Neural networks,* vol. 5, no. 2, pp.241-259, 1992.

[177] G. Shakhnarovich, T. Darrell, and P. Indyk, P., “*Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*,” *The MIT press,* 2006.

[178] G. Y. Wang, H. Yu, and D.C. Yang, “Decision table reduction based on conditional information entropy,” C*hinese journal of computers-chinese edition,* vol. 25, no. 7, pp.759-766, 2002.

[179] I. H. Witten, E. Frank, M.A. Hall, and C.J. Pal, “Data Mining: Practical machine learning tools and techniques,” *Morgan Kaufmann,* 2016.

[180] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," *Proceedings of the 17th ACM conference on Computer and communications security*, ACM, 2010, pp. 399-412.

[181] A. Majumdar, T. Debnath, S. K. Sood, and K.L. Baishnab, "Kyasanur Forest Disease Classification Framework Using Novel Extremal Optimization Tuned Neural Network in Fog Computing Environment," *Journal of medical systems*, vol. *42, no.* 10, pp.187, 2018.

[182] D. E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," *California Univ San Diego La Jolla Inst for Cognitive Science*, 1985.

[183] Katanforoosh and Kunin, "Initializing Neural Networks," *deeplearning.ai*., 2018.

[184] X. Glorot, and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249-256.

[185] A. Géron A., "Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems," *O'Reilly Media, Inc.*, 2017.

[186] J. Han, and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," *In International Workshop on Artificial Neural Networks, Springer, Berlin, Heidelberg*, June 7, 1995, pp. 195-201.

[187] D. Mishkin, andD, J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[188] V. Nair, and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807-814.

[189] D.A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[190] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[191] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[192] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015.

[193] Y. Wang, N.G. Polson, and V.O. Sokolov, "Scalable Data Augmentation for Deep Learning," *arXiv preprint arXiv:1903.09668*, 2019.

[194] D.P. Kingma, and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2012.

[195] J. Zou, M. Huss, A. Abid, P. Mohammadi, A. Torkamani, and A. Telenti, "A primer on deep learning in genomics," *Nature genetics*, p.1., 2018.

[196] B. Prin, and N. Kashyap, "Using Apache Spark with TensorFlow on Google Cloud Platform," 2017. [Online]. Available: https://cloud.google.com/blog/products/gcp/using-apache-spark-with-tensorflow-on-google-cloud-platform. Accessed: March 18, 2019.

List of Publications

1) D. Gupta, and R. Rani, "Big Data Framework for Zero-Day Malware Detection," *Cybernetics and Systems*, vol. 49, no. 2, pp.103-121, 2018.

[*SCIE Indexed, IF: 1.681*]

2) D. Gupta, and R. Rani, "A Study of Big Data Evolution and Research Challenges," *Journal of Information Science,* vol. 45, no.3, pp. 322-340, 2019.

[*SCIE Indexed, IF: 2.237*]

3) D. Gupta, and R. Rani, "Improving Malware Detection using Ensemble Learning and Big Data," *Computers & Electrical Engineering,* vol. 86, pp.106729, 2020.

[*SCIE Indexed, IF: 2.663*]

4) D. Gupta, and R. Rani, "Understanding the Trends in Evolution and Threat Landscape of Windows Malware in the Age of Big Data," *Cybernetics and Systems.*

[*Communicated: February, 2018; Status: Under Review*]

5) D. Gupta, and R. Rani, "Big Data Security Analytics - A Road towards Cyber Threat Intelligence," *Applied Artificial Intelligence.*

[*Communicated: March, 2018; Status: Under Review*]

6) D. Gupta, and R. Rani, "Malware Detection using Big Data and Deep Neural Network," *Engineering Applications in Artificial Intelligence.*

[*Communicated: May, 2019*]