8-22-2023 10:30 AM

Predicting Network Failures with AI Techniques

Chandrika Saha, *The University of Western Ontario*

Supervisor: Haque, Anwar, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

Network failure is the unintentional interruption of internet services, resulting in widespread client frustration. It is especially true for time-sensitive services in the healthcare industry, smart grid control, and mobility control, among others. In addition, the COVID-19 pandemic has compelled many businesses to operate remotely, making uninterrupted internet access essential. Moreover, Internet Service Providers (ISPs) lose millions of dollars annually due to network failure, which has a negative impact on their businesses. Currently, redundant network equipment is used as a restoration technique to resolve this issue of network failure. This technique requires a strategy for failure identification and prediction to run faultlessly and without delay. However, we lack a suitable generic network failure identification and prediction system due to the unavailability of publicly accessible failure data. This study simulates network traffic to gather failure data based on a general network failure guideline. Furthermore, various state-of-the-art Machine Learning and Deep Learning methods were applied to the generated data. Notably, our proposed Deep Learning model for failure identification provides accuracy, precision, recall, and F1 scores in the range of 97% to 99% for three different demonstration networks. Additionally, our proposed Long Short Term Memory model gives low root mean square error rates of 0.9751 for failure prediction.

Keywords

Network Failure, Network Failure Detection, Network Availability, Network Traffic Simulation, Machine Learning, and Deep Learning.

Summary for Lay Audience

Internet services may be interrupted for various reasons, such as device failure, connection issues, natural disasters, etc. When a network error inadvertently disrupts internet services, customers become frustrated. Especially in a medical emergency, personal safety, or transportation-related emergency, a lack of network connectivity can be catastrophic. In addition, since the outbreak of the COVID-19 pandemic, many businesses require a constant internet connection. In reality, every aspect of our lives depends on the Internet, including emergency needs, banking, entertainment, healthcare, socializing, and creative work. The Internet is now such an integral part of our lives that its absence will be immediately noticeable.

Additionally, Internet Service Providers (ISPs) are impacted by service interruptions from a business perspective, as they lose millions annually due to network failure. As a result, ISPs utilize redundant network equipment as backups if a network device fails. However, the disadvantage of this approach is that it requires time to identify the type of failure and then fix the problem with an appropriate backup. Constructing a system that can predict the type of failure occurring next to using backups effectively would be preferable. But building this type of system will necessitate a record of past network failures to determine what types of failures may occur in the future. Nevertheless, the absence of publicly accessible failure data is responsible for lacking a suitable general network failure identification and prediction system.

This research simulates network traffic to obtain failure data based on a generic failure guideline to address this issue. Furthermore, cutting-edge Artificial Intelligence (AI) techniques on that data yield promising results.

Acknowledgment

I am writing to express my sincere gratitude to my supervisor, *Dr. Anwar Haque*, for his unwavering support and encouragement throughout this research. His crucial guidelines and feedback at every research step have been instrumental in shaping and refining the study. His impeccable domain knowledge in the Computer Networking field has been an indispensable part of the design and development of this study.

I would also like to express my gratitude to my family. I thank my husband, *Aparup Kar*, for his constant encouragement and emotional support while completing this work. I want to express my wholehearted gratitude to my mother, *Tripti Bose*, and my father, *Basudev Saha*, for always believing in my abilities and pushing me to reach my fullest potential. My brother, *Arnab Saha*, thank you for always challenging me and helping me be the best version of myself. I am also grateful to my friends for being a listening ear and a source of encouragement during difficult circumstances. I could not have done it without you.

Once again, I would like to express my heartfelt appreciation to my supervisor, *Dr. Anwar Haque*, for his constant support and inspiration. I could not have done it without your constructive feedback and profound insight.

Table of Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

Network failure is the complete or partial failure of network hardware or software components that causes service disruptions. It is a significant concern for network service providers because a mere 1 hour of network failure affecting only 100 customers can cost them hundreds of thousands of dollars [1]. Clients are impacted because they are receiving inadequate service due to network failures. Moreover, it becomes crucial when the service is time sensitive.

Conventionally, redundant network equipment is a probable solution to address network failure [2]. These software or hardware backups serve not only in the event of a network outage but also in the case of routine software or hardware upgrades. Nevertheless, switching to a backup path as soon as a problem is detected is crucial for these fault tolerance measures to be effective. Since numerous time-critical services in healthcare, mobility control, real-time media, smart grid control, and motion control require uninterrupted internet services [3], we need a system that significantly reduces network downtime and provides excellent availability. Therefore, an appropriate failure identification and prediction model is essential to prepare our redundant network resources for potential failure.

Some conventions are utilized to ensure redundancy. For example, mesh connections among routers divert the affected network flow to an unaffected link. Additional routers and network paths are also maintained. Moreover, additional processors, memory, line cards, and links are also used to ensure redundancy [2].

If we explore further, network failure identification and prediction fall in the category of 'Availability' of the three triads of network security- CIA (Confidentiality, Integrity, and Availability) [4]. Keeping the data encrypted to render it unreadable to third parties (Confidentiality) and preventing tampering with the data (Integrity) will be ineffective if we cannot deliver the data (Availability), especially for time-sensitive operations. The time required to discover, identify, and repair a failure directly impacts the network's availability. *Equation 1.1* demonstrates the relationship between availability and failure [2]. In *Equation 1.1*, if we have an MTBF (Mean Time Between Failure) of *0*, it implies that the system is

experiencing failures so frequently that there is no operational time between failures. Again, in the case of MTTR (Mean Time to Repair) being *0*, we have a system that gets repaired instantly which is an idealized scenario and may not reflect real-world situations. Additionally, if we have *0* for both MTBF and MTTR, we have a unique scenario where failures keep occurring and getting repaired instantaneously. Therefore, *Equation 1.1* considers the realistic assumption of both MTBF and MTTR being a non-zero positive integer. According to this equation, increasing the MTBF and decreasing the MTTR will increase the network's overall availability. A reliable network failure identification and prediction model making the best use of redundant equipment is required to guarantee both conditions.

$$Availability = \frac{MTBF}{(MTBF + MTTR)} \quad if\ MTBF > 0\ and\ MTTR > 0 \qquad \textit{Equation 1.1}$$

In order to prevent network failures and guarantee maximum availability, we must be aware of their potential causes. Power outages, natural disasters, hardware or software failure, and configuration errors are the most common causes of outages. According to a reliability study conducted by the University of Michigan, the leading causes of downtime were router software or hardware upgrade related issues (36%), link failure (32%), router failure (23%), and other reasons (9%) [1], [2]. These factors must be considered to design an effective failure prediction model and effectively deploy redundant elements.

The discussion above shows that maintaining internet availability is vital for time-critical network solutions and the ISP's financial interests. This study aims to develop a comprehensive framework for network failure identification and prediction to address this problem based on a network failure guideline [2].

1.2 Motivation

Roger's significant telecommunications outage on the early morning of July 8, 2022 [5] demonstrates the significance of this network failure issue. This daylong outage across Canada has left a quarter of the population (12 million) without internet access, severely impacting their daily lives. Emergency and payment services were also unavailable with data and calling services. This incident emphasized the susceptibility of industry, financial institutions, and healthcare systems to a network failure, rendering the loss of productivity, inability to work, missed meetings, opportunities, and contracts a minor pitfall. Rogers required excessive time

to identify the cause of the failure, which turned out to be a software update of a router. Overall, this highlighted the need for a comprehensive failure prediction tool.

Maintaining a five-nine availability (99.999 percent) with 5.256 minutes of annual downtime is the gold standard, particularly for mission-critical communication [6]. Even with redundant equipment, the time to identify and repair network failure persists due to complex failures, manual intervention, and validation of the failure [2]. Redundancy reduces downtime but does not eliminate the intricacies of diagnosing and addressing network issues. A reliable failure identification and prediction model can help us in this regard to maintain quality and improve performance by making the best use of redundant network equipment. Being aware of failures caused by natural disasters, power outages, and system maintenance enables us to use redundant resources effectively and provide uninterrupted service. A failure prediction model will save millions of dollars that ISPs (Internet Service Providers) must lose due to downtimes.

As discussed, the significance of a failure prediction system is paramount for time-critical network services. It is crucial to have a framework for a network failure prediction system to maintain excellent network availability and minimal loss of ISPs.

1.3 Challenges

Machine Learning (ML) techniques must be applied to a collection of failure data to identify potential failure patterns. The primary issue at hand is the requirement for publicly accessible failure data. Network providers must be willing to make their failure information public. Even if we want to generate a synthetic dataset with a demonstration network, it is difficult because there are many parameters to consider. In addition, the simulated data should provide us with failure traces that matches the pattern of real-world failure data. Furthermore, even if we could obtain network traffic data displaying availability and failure traces, the proportion of failure data would be significantly lower than that of regular traffic, making it more challenging for an ML algorithm to identify failure patterns in traffic. These obstacles are described in detail in the following subsections.

1.3.1 Unavailability of failure data

Examining the case studies of significant network failures over the past few years will reveal the general dearth of data on network failures. Section 2.1 of Chapter 2 of this paper

analyses some of these network failure incidents in detail. However, examining past massive network failures, we find that most of their causes were never disclosed. Again, some of the failures in the past fall into specific categories, such as software or hardware failure, misconfiguration, power supply failure, and fiber cut (Chapter 2, Section 2.1). As there are a few prominent types of failure, attempting failure prediction with sufficient data is possible.

However, some outdated public failure datasets were found after an extensive search (Chapter 3, Section 3.2). Most of these datasets consist of CPU and node availability and ping logs. Furthermore, these datasets were mainly used to analyze the viability of specific configurations for grid computing. All these log datasets were from the years 2000–2005. Other datasets include manually recorded software (1996–2005) and hardware (2003–2007) failure logs of some facilities. These datasets are outdated and were not designed for predictive analysis. In addition, they do not account for all possible failure categories.

1.3.2 Challenges of generating synthetic data

Generating synthetic data to solve the problem of the lack of actual data is a challenging endeavor. We must consider the intricate details of the real-world scenario. For our case, generating network failure data will require us to capture the complexity of actual network architecture. There are numerous interdependent parameters to consider. Designing a system that produces data while keeping the relationship between the parameters intact requires careful consideration of the properties of the elements of the network. Again, the generated dataset must follow the statistical distribution of the real-world data. Overall, capturing the real world's randomness while maintaining the dependencies between variables and their resemblance to actual distribution is incredibly challenging.

1.3.2 Unbalanced nature of failure data

By default, network failure records are fewer than regular traffic records in a network traffic flow. Additionally, some failures show randomness, such as those resulting from natural disasters or human error. In addition, specific failures (equipment failure, misconfiguration) occur much more frequently than others (power supply failure). For these reasons, it is difficult to classify network failure even with a dataset. Evaluating various ML algorithms and identifying the optimal one is essential to classify failure data correctly. Also, it is necessary to

balance the data so that the classifier can accurately identify the minority categories. Oversampling minority classes or under sampling majority classes is a possible method for balancing failure data. As an alternative method, it is also possible to generate synthetic data using deep learning techniques. Finding a suitable ML model for failure categorization, even with a dataset, is extremely difficult.

1.4 Thesis Contribution

This research employs the following methods to address the obstacles cited in the previous section regarding the construction of a network failure model:

- A general failure guideline is evaluated and applied to three different demo network topologies of varying sizes [2]. Various features are considered, such as the discontinuation of production, port utilisations per node, power supply, memory, and card risk factors.

- With these demos, sample traffic data are simulated. This network traffic data contains the node utilization rates and risk factors with their availability status: regular traffic or port, card, CPU, memory, power supply, location, OS, and misconfiguration error. To the best of our knowledge, this is the only existing generic network failure dataset.

- This generated dataset is subsequently trained with multiple cutting-edge ML algorithms and deep learning techniques to form a comprehensive study and determine the most suitable model for network failure prediction with proper hyperparameter tuning.

- Using deep learning techniques with hyperparameter tuning the sequence of each node's availability state is analyzed in order to predict future failure events.

1.5 Thesis Outline

The remaining sections of the thesis are structured as follows: In Chapter 2, the components of a network, recent significant network failures, and AI-based prediction techniques are described. The third chapter describes related research on failure prediction and comprehensively analyzes publicly accessible network failure logs. Chapter 4 analyzes the

entire procedure for generating the proposed failure dataset in-depth, along with the experimental specifics of the AI techniques. The experimental outcomes are described and illustrated in Chapter 5. The sixth chapter of the thesis concludes with possible future directions.

Chapter 2

2 Background

This Chapter describes the concepts and statistics associated with network failure. Section 2.1 gives an idea of the building blocks of a network and the type of failures that can happen. Section 2.2 summarizes past catastrophic network failures, including their duration and impact. In addition, the final section describes the AI techniques considered for this investigation.

2.1 Network Building Blocks

Internet Service Providers (ISP) provide customers with internet services worldwide. To distribute the Internet to their clients, ISPs use physical infrastructure, network equipment, and software [7]. As physical infrastructure, they use optical fiber cables or wireless networks. To route data to their customers, they use routers and switches. These routers or nodes are interconnected to maintain the global flow of network traffic.

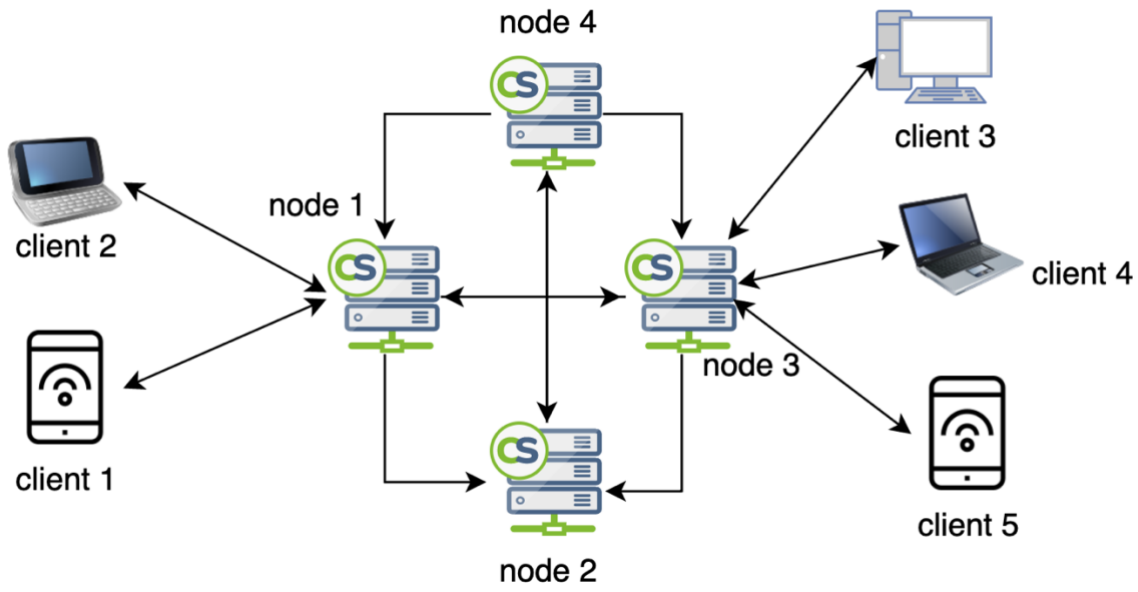

Figure 2. 1: Simplified view of a network architecture

Figure 2.1 demonstrates a simplified network architecture. The link between routers or nodes is known as a *network link*. This *link* is used to transfer data between nodes. For instance, if *client 1* wishes to send a text message to *client 4*, they may transmit it to *node 1* first. It can be forwarded from *node 1* to *node 3* and then to *client 4*. In this context, the *network traffic* is

the text being transmitted. The *software component* of the node determines how data will be transferred between nodes. A node's number of outgoing connections is known as its *nodal degree* [8]. For example, *node 1* has a *nodal degree* of *2*, while *node 2* has a *nodal degree* of *1*. The following subsection describes the functional component of a node in greater detail.

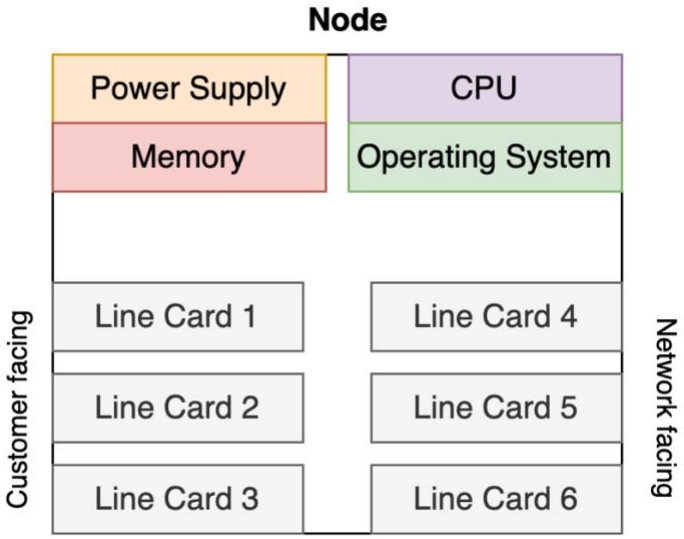

Figure 2. 2: Internal structure of a node

2.1.1 Nodes

A *node* or *router* in a network is a point where data processing and transmission occur (*Figure 2.2*). Every *node* has a manufacturer-provided lifetime. Each *node* is a functional network component consisting of a power supply, memory, CPU, and an Operating System. Each *node* has several line cards with several ports [7]. The functions of each of these components are as follows:

- **Power supply:** This is the node's electrical outlet, which supplies the node with electricity. Each node requires an adequate quantity of power for the device to function. A faulty power supply will entirely shut down the node and may cause damage to the node's other components.
- **CPU (Central Processing Unit):** A node's CPU is responsible for processing and routing data and maintaining scheduled tasks. Among the duties that can be scheduled are security checks and network protocol management. A CPU failure will halt all processing-related jobs and render the node ineffective.

- **Memory:** A node's memory refers to its storage capacity. This memory stores data prior to processing or transmission. Frequently, a *memory buffer* is used to store additional data beyond the processing capacity of the node to reduce congestion.
- **Operating System (OS)**: This software component of a node manages the resources and activities of a node. It is responsible for managing each aspect of a node, including data transfer, managing protocols, and applying security measures, among others.
- **Line Card:** This is one of the hardware components of a node that acts as an adapter. A line card provides interfacing ports that connect the node to other network nodes. A *port* is an interface that facilitates the transmission of data between nodes. Each node has a succession of client-facing cards connecting to other clients. In addition, the side confronting the network is connected to other nodes (Figure 2).

2.1.2 Failure Categories: Cisco Guidelines [2]

In [2], Cisco Press provided a guideline based on a one-year study by the University of Michigan on the availability study of a regional ISP. Based on the survey, network failures can broadly be divided into hardware and software failures. Within hardware failure, there are subcategories like power supply, memory, CPU, card, port, link failures, and natural disasters. Among software failures, we have OS failures and misconfiguration failures. However, the percentages of each failure category's frequency and the contributing factors are also provided in greater detail.

Most network failures (32%) are due to *link failures*, primarily caused by *fiber cuts* and *configuration errors* (*Table 2.1*). Again, *operating system failure* (18%) and *configuration failure* (18%) are the second-most common causes. *Upgrade* and *maintenance* issues are the primary causes of *OS failures*. *Card* and *port* failures follow with a 10% incidence rate each. These occurrences result from either *high utilization* rates or the device wearing out due to extended use (*manufacturing discontinuation*). Then there are *natural disasters* and other unpredictable and uncontrollable causes (9%). Last but not least, we have *power supply failures* (1%), *memory failures* (1%), *and CPU failures* (1%), which occur due to *high utilization* and *overuse*. In addition, all of these causes of network failure can be confirmed through the case studies described in the following section of this chapter.

Table 2. 1: Failure categories, Percentages, and Contributing factors.

Failure Categories	Percentage	Factors	
Power supply failure	1%	Manufactured discontinued	-
Memory failure	1%	Very high utilization	Manufactured discontinued
CPU failure	1%	Very high utilization	Manufactured discontinued
Card failure	10%	Very high utilization	Manufactured discontinued
Port failure	10%	Very high utilization	Manufactured discontinued
Link failure	32%	Fiber cut- construction	Configuration error
Natural disasters and others	9%	-	-
OS failure	18%	Upgrade	-
Misconfiguration failure	18%	Config- fat finger	-

2.2 Network Failure Case Studies

Several significant network failure incidents from 2015 to 2021 are examined to understand the network failure's impact thoroughly. This section outlines the root cause of each incident and its impact on real-life scenarios.

2.2.1 Unknown/ Undisclosed Network failures

Rogers Telecommunications experienced similar outages, as mentioned in *Chapter 1*, in 2015 and 2019. In July 2015, *Rogers and Fido* customers in certain provinces of Canada experienced intermittent disruptions of calls, texts, and data services [9], [10]. During the 2019 nationwide outage, *Rogers's* customers experienced disruptions to their voice call services [11], [12]. During this outage, even 911 emergency calls were not possible. However, the reasons behind any of these outages were not disclosed.

On February 16, 2016, *Comcast* cable and Internet Service Provider customers experienced an outage [13]. Due to this 90-minute outage beginning at 10 a.m., customers could not view on-demand videos, broadcasts, or local cable TV channels. Customers were subsequently compensated for the service interruption they endured [14]. It was unknown what caused this outage.

On August 6, 2020, beginning at 12:10 p.m., a significant portion of Ontario and specific locations in Quebec experienced a network outage. *Bell* and *Telus* customers were the

most affected by this outage. The lack of internet, television, mobile phone, and landline services has caused widespread customer frustration. The blackout lasted slightly less than an hour. However, the cause of the outage remained unknown [15].

The United States and parts of Western Europe experienced a widespread outage on Sunday morning, August 30, 2020 [16]. A *CenturyLink* equipment malfunction caused this outage. A faulty flowspec rule prevented BGP (Border Gateway Protocol) from being broadcast across the network. Later, an IP NoC global configuration change halted the offending flowspec. However, the cause of the flowspec's activation remained unknown. We can only speculate that a misconfiguration or cyberattack is the reason. This outage affected Discord, Feedly, Xbox Live, Hulu, and other data centers using the ISP [17]. It took *CenturyLink* four hours to identify and resolve the issue.

The northeast coast of the United States remained without internet access on January 26, 2021, due to the outage of *Verizon Wireless* and its fiber optic network *Fio*. Clients reported outages in other services, including Gmail, Google, Slack, Zoom, AWS, and Hulu [18]. During this post-Covid period, many clients could not join their remote work, and students were absent from online classes. However, the power outage began at 11 a.m. and lasted over an hour [19]. The incident's root cause was not disclosed. This incident demonstrates that even if redundancies are installed, a large-scale outage could render them useless if traffic levels are high. This is especially true if most individuals work from home or attend online school.

2.2.2 Equipment failure

On December 27, 2018, *CenturyLink* experienced one of the most significant outages in its history, affecting 22 million customers in 39 U.S. states [20]. Clients could not access the Internet or emergency 911 services for 37 hours [21]. A total of 12 million calls dropped as a result of this outage. However, a malfunctioning piece of equipment that caused misconfiguration in the network nodes was the cause of this outage. One of the switching modules began generating improperly formatted packets. Due to the absence of suitable filters, these packets propagated to all other connecting nodes. This incident caused a power outage by creating an infinite feedback loop that consumed all available processing power. Due to the severity of this outage, the *FCC (Federal Communications Commission)* later conducted an investigation.

In another incident, expired certificates caused an outage in *Ericsson* equipment in December 2018. Because they used *Ericsson* software, millions of customers from 11 countries experienced poor connections [22]. Customers of *O2* in the United Kingdom and *SoftBank* in Tokyo were offline for several hours. It took *O2* over 12 hours to fully recover from the issue, while *SoftBank* customers were affected for only four hours [23]. In addition to hindering communication, outages resulting from expired certificates exposed it to third parties.

During a 40-minute outage on 1 April 2019, 780 flights were delayed or canceled across the United States [24]. Due to a software defect at *AvroData Inc*., flight planning and weight balancing became difficult, resulting in delays. Some airlines impacted by this problem include Southwest, SkyWest, and United Continental. Travelers were forced to endure lengthy airport waits and had their travel plans disrupted due to this outage.

A software flaw in a recent update caused an outage on June 8, 2021, for the Content Delivery Network (CDN) *Fastly* [25]. A customer's legitimate change of configuration command caused the bug. Due to this, 85% of their network returned errors. The New York Times, Hulu, CNN, Amazon Web Services, Twitter, Spotify, and Reddit failed to deliver services to their clients. Some of *Fastly*'s clients reverted to non-CDN mode, which could result in slower-than-usual traffic due to the lack of load balancing [26]. However, resolving this issue took approximately three hours, and it took even longer to debug and redeploy the updated code.

2.2.3 Misconfiguration Error

On 12 June 2015, *Level 3 Communications* experienced a two-hour outage that affected the United States and parts of Europe (the United Kingdom, France, and Germany) [27]. Due to this outage, Google, LinkedIn, AOL, Microsoft, and Dow Jones could not provide services. *Capital One* suffered collateral damage because *Level 3 Communications* was their primary ISP (Internet Service Provider). This outage was the result of a configuration error. Due to inadequate filtering, *Telecom Malaysia* inadvertently advertised itself as a route, which *Level 3 Communications* (one of the largest Tier 1 ISPs) accepted. As a result, network traffic flowed toward *Telecom Malaysia*, resulting in countless packet drops and service interruptions.

In another incident, an *Amazon S3* service disruption in February 2017 halted many services in northern Virginia, including Coursera, Quora, and Trello. A later investigation

revealed that this outage was caused by human error [28]. While debugging a slower-than-usual billing system, the *S3* team inadvertently shut down several vital servers, resulting in a four-hour-long outage. After accidentally shutting down the servers, restarting and bringing them back online took considerable time. This widespread outage cost 500 businesses $160 million [29].

On 2 July 2019, *CloudFlare* witnessed a global outage caused by a flaw in one Web Application Firewall (WAF) rule [30]. The erroneous code segment was a regular expression that demanded extensive backtracking, consuming excessive processing power. As a result, users could not access sites and services hosted by *CloudFlare*. However, this outage raised concerns regarding the dependability of cloud services and the necessity of a backup plan in the event of a similar failure.

2.2.4 Power supply failure

Due to an outage at the *Equinix* data center in July 2023, clients in London could not access one out of ten web pages. One of the *LINX* exchange nodes failed due to the outage, impeding twenty percent of the total network flow. A UPS power disruption of one of *Equinix*'s subsidiaries, *Telecity*, caused the failure. *Telecity* resolved the issue in 22 minutes, whereas *LINX* required more than an hour. In 2015 and 2011, similar UPS failures occurred in *Telecity* [31].

2.2.5 Fiber cut

Bell's outage on August 4, 2017 [32] affected banking services, 911 emergency calls, and even air travel. Internet, television, and landline phone services were disrupted throughout most of Atlantic Canada. The outage began before 11 a.m. and lasted until 4 p.m. Accidentally damaged fiber wires during construction caused this widespread outage. However, a lack of adequate backup also contributed to the severity and duration of the outage. *Bell Aliant* experienced a similar outage in New Brunswick in 2011.

2.3 Artificial Intelligence

Artificial intelligence (AI) is the study of training machines to perform tasks that humans can do [33]. Machine learning (ML) is a subfield of AI that learns how to perform a

given task by accumulating experience through repeated attempts [34]. ML algorithms analyze a given dataset and discover patterns to draw meaningful conclusions. Deep Learning (DL) is a subset of Machine Learning (ML) that employs computationally intensive models to discover complex patterns in given data [35]. A Recurrent Neural Network (RNN) is a deep learning technique that processes sequential data through information propagation [36]. In the context of this study, if we provide a machine learning algorithm with network failure data, it can identify a pattern in the data and predict the type of failure based on the pattern it has learned. The subsequent subsections detailed the ML and DL techniques utilized for this thesis.

2.3.1 Logistic regression

Logistic regression is a statistical model that is used for classification. Using a logistic function, this model outputs the probability that a sample is one of the possible categories [37]. This technique uses the sigmoid function as the hypothesis function, where θ is the set of learning parameters, and x is the input data (*Equation 2.1*). In this case, the hypothesis function's task is to provide a probabilistic measure of a particular data sample x_i to be of a certain category.

$$h_\theta(x) = \frac{1}{(1 + e^{-\theta^T x})} \qquad \textit{Equation 2.1}$$

The learning parameters θ are optimized with labeled training data to ensure the hypothesis function is accurate. The cost function in *Equation 2.2* is used to optimize the parameters for binary classification. The learning parameters are modified with an optimization technique to obtain the lowest cost. In *Equation 2.2*, $h_\theta(x)$ is the hypothesis function, and y is the actual category of a sample.

$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & y = 1 \\ -\log(1 - h_\theta(x)), & y = 0 \end{cases} \qquad \textit{Equation 2.2}$$

For a multi-class dataset, the softmax function is used to predict the data point category. A softmax unit outputs a list containing the probability of a particular data point for every possible type. From this list, the class with the highest value is chosen as the predicted label. And as an optimizer, sklearn's ‘SAGA’ is used, a variation of the stochastic gradient descent algorithm that uses the loss function from a random sample of data for faster convergence.

2.3.2 Naïve bias

Naive Bayes is a probabilistic algorithm that is used for classification. The model runs under the assumption that the features of the given dataset are independent of each other [38]. This model calculates the probability of a given sample based on the observed training data. The model works with the following equation:

$$P(y \mid x_1, x_2, \dots, x_n) = \frac{P(y) * P(x_1 \mid y) * P(x_2 \mid y) * \dots * P(x_n \mid y)}{P(x_1, x_2, \dots, x_n)} \qquad \textit{Equation 2.3}$$

Here we are calculating the probability of a given data sample $(x_1, x_2, \dots, x_n)$ being a category y using the prior probability of the category $(P(y))$, the conditional probabilities of feature x_i given the category is y $(P(x_i \mid y))$, and the probability of the features occurring together $P(x_1, x_2, \dots, x_n)$.

In the case of Gaussian naïve bias, the conditional probability of a feature given a category is assumed to follow a normal distribution (*Equation 2.4*).

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}} \qquad \textit{Equation 2.4}$$

Here, μ_y and σ_y^2 is the mean and standard deviation of the feature set for each class label.

2.3.3 Support Vector Machine (SVM)

Support Vector Machine is a popular supervised Machine learning algorithm used for binary and multi-class classifications [39]. SVM works by finding a hyperplane that will maximize the margin among different categories of data points. To obtain this highest margin and to define the shape and direction of the hyperplane, SVM uses support vectors (data points closest to the margin). The hinge loss function penalizes the misclassified samples to maximize the margin (*Equation 2.5*).

$$Cost(y, h(x)) = \begin{cases} 0, & y * h(x) \geq 1 \\ 1 - y * h(x), & else \end{cases} \qquad \textit{Equation 2.5}$$

SVM uses many kinds of kernels to manipulate the input data according to the problem statement. The linear kernel is one of the most used kernels that takes the dot product of input feature vectors, i.e., it is simply the linear transformation of the training data points. The formula for the linear kernel is given in *Equation 2.6*.

$$K(x_i, x_j) = x_i^T x_j$$

Equation 2.6

Here, x_i and x_j is input data points.

Another Popular kernel used with SVM is 'rbf' or Gaussian kernel, which maps the training data into a higher dimensional space. This kernel works best when the data points are not linearly separable, which for most problems with a higher number of features is true. The RBF kernel uses *Equation 2.7* where x_i and x_j are input data points and γ is a parameter used to provide a regularization effect:

$$K(x_i, x_j) = e^{-\gamma \left||x_i - x_j|\right|^2}$$

Equation 2.7

2.3.4 Decision tree

The *decision tree* is another rule-based machine learning algorithm popular for prediction and classification tasks [40]. *Figure 2.3* depicts the entire process of creating a decision tree. A decision tree first takes the entire dataset as the root node. Then it selects a feature that will provide the most information gain and partitions the input space. It then recursively does the same calculation on each decision node. The 'gini' impurity is used for this study to calculate the information gain of a particular feature. Gini impurity is calculated as follows:

$$Gini\ Impurity = 1 - \Sigma\ (p_i)^2$$

Equation 2.8

Here p_i is the probability of each class i in the dataset.

After the recursive process, we have a tree like *Figure 2.3* which is used to predict the category of a new data point.

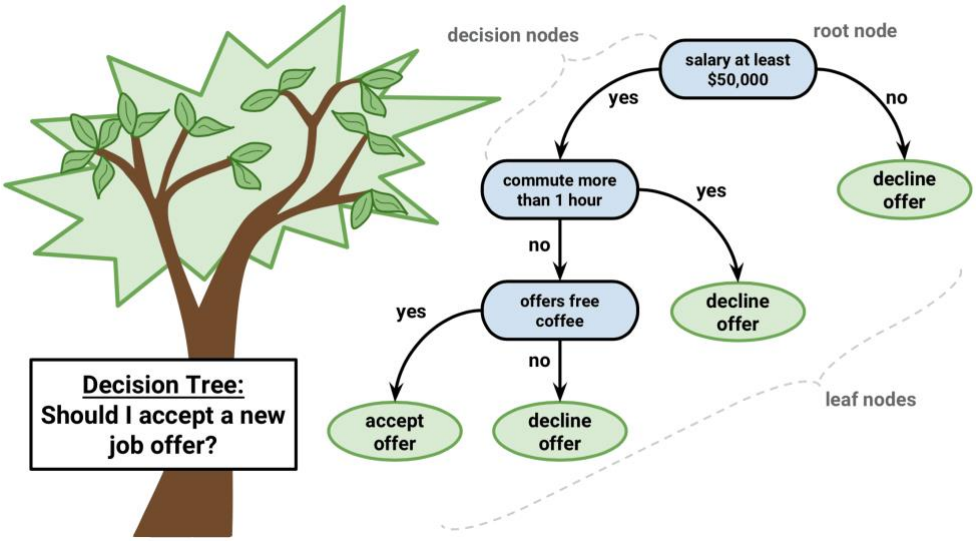

Figure 2. 3: Decision Tree (Image source: Learning Spark, 2nd Edition, figure 10–9*)*

2.3.5 Random Forest

Random forest is a variation of the *decision tree* algorithm, which uses a majority voting technique to make a generalized model [41]. The underlying concept of creating a tree based on the 'information gain' is like the *decision tree*. However, a random forest classifier works with the following steps:

1. A random subset of features is selected from the feature set.
2. *Decision tree*s are built with that section of the data.
3. Steps 1 and 2 are repeated with different feature sets, creating a forest of trees.
4. A new data point is tested by running it through all the trees in the forest and taking the majority vote to choose the category.

Random forest increases generalization ability and decreases overfitting by bringing randomness into building trees and picking categories. *Figure 2.4* depicts the overall process of random forest classification.

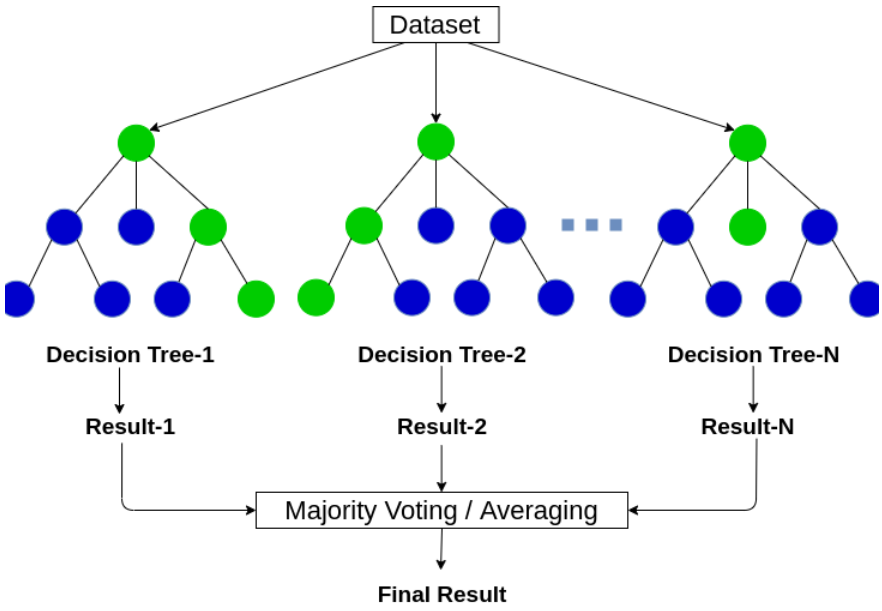

Figure 2. 4: Random Forest (Source: medium.com, The Complete Guide to Random Forests: Part 2)

2.3.6 AdaBoost

AdaBoost or *Adaptive boosting* is an incremental ensemble machine learning technique used for classification [42]. *AdaBoost* work on ensemble learning where the average of some weak models is shown to provide superior performance than one single model. The *AdaBoost* classifier uses the following steps:

1. Initially, random weights are assigned to all training samples.
2. A weak classifier is trained with the initial sample.
3. Training data is evaluated with the model.
4. Weights are penalized if misclassification is observed.
5. Steps 1-4 are repeated with a new sample set generating another weak classifier. This sample set mainly contains misclassified data points from the training set.
6. Take majority voting for all the classifiers.

This type of ensemble-based model is known to deal with large datasets while maintaining low overfitting.

2.3.7 Gradient boosting

Gradient boosting is a similar technique to *AdaBoost*. The gradient boosting algorithm also works by taking the ensemble of a few weak models [43]. In the case of *gradient boosting*, the loss function of each subsequent model is penalized with the residual errors of the previous model rather than focusing on the misclassified samples, as in *AdaBoost*. This model also provides a majority voting-based generalized solution. Gradient boosting works as follows:

1. Initialize the model with a constant value.
2. A weak classifier trained with the initial sample.
3. The residual error of the model on the training data is calculated.
4. Fit another weak learner to the residual errors and add it to the model.
5. Update the model by adding a fraction of the new learner to the previous model.
6. Repeat steps 2-5 until the desired accuracy is achieved.

2.3.8 Multi-Layer Perceptron (MLP)

A *Multi-Layer Perceptron (MLP)* Classifier is an artificial neural network commonly used for classification tasks [44]. A feedforward neural network consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of several neurons that are connected to the neurons in the adjacent layers. The neurons use activation functions to transform the input data and pass the information forward to the next layer until the output is produced. *Figure 2.5* depicts an MLP classifier.

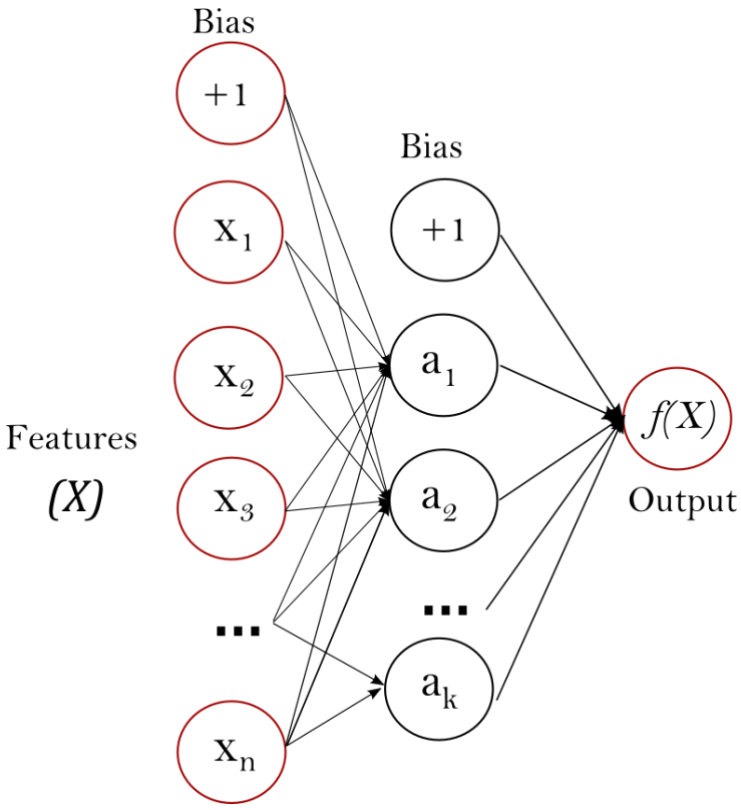

Figure 2. 5: Multi-Layer Perceptron - MLP (Source sci-kit learn)

2.3.9 Quadratic discriminant analysis

Quadratic Discriminant Analysis (QDA) is a classification algorithm used to model each class's probability distribution [45]. The goal is to find the class that maximizes the posterior probability given the input features. Bayes' theorem is used to compute the posterior probability of each class. The probability of sample x belonging to a class C is given by:

$$P(C|x) = P(x|C) * \frac{P(C)}{P(x)} \qquad \textit{Equation 2.9}$$

where $P(x|C)$ is the probability of observing x given that it belongs to class C, $P(C)$ is the prior probability of class C, and $P(x)$ is the probability of observing x.

QDA assumes that the input features follow a normal distribution with each class, and therefore the probability distribution of each class can be modeled as a multivariate Gaussian distribution. The mean vector and covariance matrix of each class is estimated from the training data.

The posterior probability of each class is computed to classify a new sample, and the class with the highest probability is chosen as the predicted class. The decision boundary between two classes is quadratic, meaning it can be curved.

2.3.10 Deep neural network

A *Deep Neural Network* is an Artificial Neural Network (ANN) containing multiple layers of interconnected neurons [46]. Each node serves as a computational point containing a set of learnable parameters. DNN is designed to model complex patterns and relationships in data. Unlike traditional shallow neural networks, DNNs can have many hidden layers between the input and output layers, allowing them to learn hierarchical features and make much more accurate predictions. *Figure 2.6* depicts a Deep Neural Network (DNN).

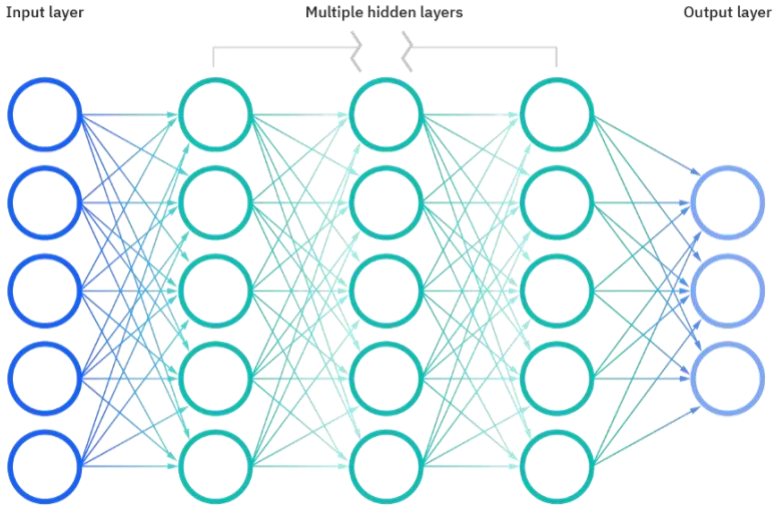

Figure 2. 6*:* Deep Neural Network (source: ibm.com)

Every learning parameter of a neural network is first initialized with a random weight. Each neuron in a DNN receives inputs from the neurons in the previous layer, performs a weighted sum of the inputs, and applies an activation function to the result. This activation function acts as a method to add non-linearity to the otherwise linear data. The output of the activation function is then passed to the neurons in the next layer as input.

$$z_j = \sum_i (w_i * x_i) + b_j$$
$$a_j = f(z_j)$$

Equation 2.9

For a neuron in layer j , with input x_i, and weights w_j, and an added bias value of b_j, the output a_j with activation function f, DNN follows *Equation 2.9*. After one such iteration, the DNN model predicts the output of a given sample as being one of the possible categories. This predicted output is compared with the actual output and a loss value is calculated. Based on this loss value, the weights of the connections between neurons are learned during training using a variant of gradient descent optimization algorithms. This process of learning weights is called *backpropagation*. For this study, *Adam* (Adaptive Moment Estimation) is used as the optimization technique [75]. *Adam* optimizer is a gradient-based optimizer that uses the concept of bias-corrected exponentially weighted average. *Adam* fits the parameters using a weighted average of derivatives of the loss function with respect to parameters.

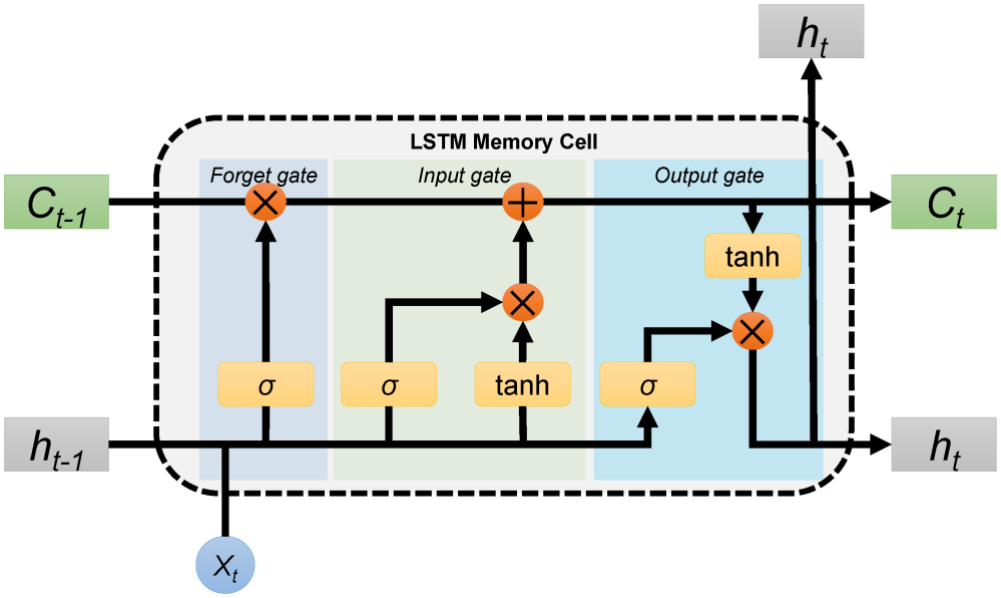

Figure 2. 7: Long Short-Term Memory Block Diagram [47]

2.3.11 Recurrent Neural Network

Long Short-Term Memory (LSTM) is a type of *Recurrent Neural Network* (RNN) that works on sequence prediction by understanding the long-term dependencies of the given sequence [48]. *Figure 2.7* depicts a block diagram of an LSTM. As we can see, an *LSTM cell* consists of three types of gates. The *forget gate* with the *sigmoid* activation function decides which information to disregard. The *input gate* with the *tanh* and *sigmoid* activation function decides which information to keep from that cell state. Moreover, the *output gate* with the *tanh* and *sigmoid* function decides which information to pass to the next state and output based on

the data from forget and input gate. Usually, several such LSTM cells are stacked to create a model.

Chapter 3

3 Literature Review

This chapter thoroughly evaluates the existing literature on network failure identification and prediction. The purpose of this analysis is to identify key themes in the existing literature and identify potential research gaps. Following are descriptions of the relevant researches in this field and the available failure datasets.

3.1 Related Works

Labovitz et al. [49] of the University of Michigan used Border Gateway Protocol (BGP) data from a large regional internet service provider to identify the leading causes of network failure in their study on internet stability. They concluded that software or hardware malfunctions, fiber cuts or network congestion, software or hardware upgrades, configuration errors, and other errors are the leading causes of network failure. In addition, they discovered that the Internet has significantly more availability problems than the telephony network.

Hayford-Acquah et al. [50] introduced a survey-based investigation into the cause of fiber cuts in Western and Central Ghana. They concluded that government road projects and private construction negligence were primarily responsible for these fiber cuts. Several short-term and long-term solutions were proposed to address these problems, such as raising awareness by educating respective parties on the repercussions of fiber damage, law enforcement, and many more. In a separate related study, Nyarko-Boateng et al. [51] utilized linear regression to predict the location of faults in optical fiber accurately. This research aimed to close the gap between the optical time-domain reflectometer (OTDR) output and the actual fault location.

Wang et al. proposed a failure prediction model for an optical network using SVM (Support Vector Machine) and DES (Double Exponential Smoothing) algorithms [52]. They collected WDM (Wavelength Division Multiplexing) data of a telecommunication operator on the control plane and selected a set of features that they used to train the SVM model. DES is then used to update the prediction curve for the equipment. With this setup, they successfully built a model that can predict which equipment can fail in the future so that they can back up the data and use a different link to continue communication.

In their paper, Zhang et al. proposed an LSTM-based deep learning model to predict failures from heterogenous streamed console log data [53]. Using the console data collected from a Web Server Cluster (WSC) and Mailer Server Cluster (MSC), they grouped similar logs together and then extracted the log cluster's format. Later, sequential features over time are extracted and fed into an LSTM model that predicts the likelihood of failure in the future. They demonstrated that LSTM models outperformed other cutting-edge Machine learning algorithms, such as Support Vector Machine (SVM) and random forest.

In their paper, Zhong et al. [54] analyzed network failure using 14 months of alarm log data collected from metropolitan area networks. After collecting the log data, separate feature sets for network system and equipment failures were developed. To create feature sets, they utilized several distinct types of alarms, the mean-variance of each type of alarm, and the frequency of failure occurring during each time interval. They use these characteristics to predict system and equipment failure probabilities using various cutting-edge machine-learning algorithms (RIPPER, Bayes Net, and Random Forest). RIPPER performed better for both system and equipment failure prediction among these algorithms.

Using simulated log data, Ji et al. constructed a classification model that identifies failure and non-failure data categories. The paper needs to describe how the log data were generated [55]. However, they labeled the cleaned heterogeneous log data using a sliding window technique. Subsequently, an embedding layer was applied to convert these labeled samples into numerical values. These numbers were utilized as input values for a convolutional neural network. Finally, the output values indicate whether a given network flow example represents a failure.

Javadi et al. introduced a Failure Trace Archive (FTA) in [56]. In this archive, they have collected several failure traces and unified them in a singular format. Most traces are log data collected from parallel and distributed systems. Based on the statistical properties of the trace, they determined that the Gamma, Lognormal, and Weibull distributions are the best fit for the majority of availability and unavailability distributions. They underlined the need for publicly available failure traces and emphasized further studies on general distributed systems.

A method for predicting network congestion is proposed in [57]. This study employed a Bayesian network to analyze the throughput data gathered from a small demonstration network. This model attempts to prevent network failure by anticipating when congestion may

occur. In another study, Xu et al. [58] proposed an RNN model for assessing the health status of hard drives. Instead of categorizing the status of discs as failing or not failing, they provided a more nuanced status with six health levels.

3.2 Available datasets

Several public availability traces are described in this section. Most of these datasets are log data found after an exhaustive search for publicly accessible failure records. However, these datasets appear outdated and unsuitable for failure prediction systems.

3.2.1 LANL

LANL05 is a dataset containing the Los Alamos National Laboratory's (LANL) manually recorded failure data from 1996 to 2005 [59]. There are 23,000 failure records for the 22 production computing systems at LANL, which contain 4,750 nodes. Each failure record contains various information, including the number of processors, date of node installation, date of production, date of decommission, CPU and memory type, number of links, the node's purpose, the time the failure began, and the time the problem was resolved. In addition, the failure type was recorded as hardware failure, human error, software failure, or an unknown category. The dataset analysis determined that system failure rates and repair times vary significantly. In addition, the failure rate is proportional to the workload. This dataset contains information on the type of failure and parameters associated with the failure.

3.2.2 G5k06

G5k06 is a collection of availability data for a large-scale grid structure known as Grid5000, which consists of nine sites in France with fifteen clusters and two thousand five hundred processors [60]. This dataset contains log data for every node in the grid and information about their availability at a given time. The information is gathered between mid-May 2005 and mid-November 2006. However, the analysis of this dataset reveals that resource availability in a grid system varies significantly. Automated resource scheduling techniques are significantly more effective than human intervention in resolving this issue [61]. It is essential to monitor and forecast the resource availability of a grid system to ensure this.

3.2.3 Microsoft99

The Microsoft99 dataset contains ping data collected from 51,663 desktop PCs over 35 days. The initial objectives of the study were to examine the viability of a serverless distributed system in which client computers do not assume mutual trust [62]. This dataset contains 52 text files, each containing hourly machine availability information. Each file contains 1000 lines, representing a one-hour snapshot of a specific machine. The snapshot shows the machine's availability as 0 (unavailable) or 1 (available) for each hour. However, they concluded that their proposed serverless distributed system would work well with any existing desktop PC network, mainly if additional storage space is installed.

3.2.4 Websites02

The Websites02 dataset was developed at Carnegie Mellon University as part of an availability model of the data distribution system. The dataset contains the ping log of a web page. Each file line contains the server's availability information and a timestamp. Additionally, different types of unavailability were recorded. Pings were transmitted every 10 minutes. Eight months of data were collected between 2001 and 2002 [63].

3.2.5 Overnet

Overnet is a dataset designed while studying availability for peer-to-peer systems. This dataset is a log of availability information for 3000 hosts checked every 20 minutes a week since January 21, 2003. Each data line in these files contains the host identifier, the host's IP address, and information regarding the host's unavailability [64].

3.2.6 ND07CPU

The ND07CPU dataset contains information on CPU load data for the Condor resource pool at the University of Notre Dame. There is a total of four months of data from early 2007. Each record contains CPU loads measured every 16 minutes, along with a timestamp and the state of the CPU. The state of the CPU is one of the following: available, idle, CPU load over 50%, and unavailable. In [36], the original timestamped CPU data were assigned to one of four states to investigate the viability of using this information to predict workload and distribute resources

more effectively. This investigation revealed the possibility of an online predictive model for the availability behavior of a multi-grid system.

3.2.7 SKYPE

The SKYPE dataset is created for an experimental study on the Skype protocol [65]. Over 82 million data points were collected from application-level pings in a Skype super-peer network. Data regarding the nature of traffic, the number of online clients, and the number of super nodes were collected from 1 September 2005 to 14 January 2006. This study aimed to examine Skype's "black box" to determine how it differs from other VoIP systems. They concluded that Skype was fundamentally distinct from other VoIP protocols and warranted further investigation.

3.2.8 SAT

The SAT dataset consists of 230,000 host availability records from the large-scale distributed system SETI@home [66]. This data set contains CPU availability information for multiple heterogeneous SETI@home hosts. It contains data for approximately two years, from 1 April 2007 to 1 January 2009. The primary objective of the dataset was to determine if a particular distribution exists among the hosts of a distributed network. They concluded that approximately 34% of hosts exhibit random behavior. Different distributions, including gamma, log-normal, and Weibull, can represent other hosts' availability.

3.2.9 PNNL

The PNNL dataset contains time-stamped hardware failure data from a high-performing computing testbed, namely, the Pacific Northwest National Laboratory (PNNL) [67]. The testbed consists of 980 nodes with an Itanium-2 processor. Overall, information on failures from November 2003 to February 2007 was compiled. Each entry contains the failure time, the node identifier, the failed node component, and the corrective action. The hard disk was the most frequently involved component in failures, then memory, graphics cards, operating system failures, and other unknown causes. It was necessary to replace the failed component to recover from most failures. In some instances, restarting or resetting the device resolved the issue.

3.2.10 UCB

UCB is a trace of the activities of a workstation cluster used by a CAD (Computer Aided Design) group at the University of California, Berkeley [68]. Keyboard, mouse, disk, memory, and CPU activities were recorded every 2 seconds for February and March 1994. This trace determined whether a workstation could handle a combined parallel and sequential workload. Overall, they demonstrated that fifty percent of the workstations in a network could adapt to the configuration of a parallel and sequential workload.

3.2.11 SDSC, LRI, DEUG

SDSC is a dataset that contains CPU availability information for a desktop grid running Entropia's DCGrid software at the San Diego Supercomputer Centre (SDSC) [69]. The dataset contains information regarding the availability of 275 hosts for 28 days between August and October 2003. This study aimed to investigate the characteristics and efficacy of a desktop grid with tasks of varying granularity. In addition, they proposed a quantitative measure of the desktop grid and dedicated cluster equivalence. Later, additional traces from the University of Paris South were added to the study [70]. The trace LRI contains CPU availability information for a cluster of forty hosts utilized by scientific calculation groups. The other trace, DEUG, is an availability trace of forty hosts in a classroom. These hosts of DEUG trace were primarily utilized during the week by first-year students. Both traces utilized the open-source grid software XtremWeb.

The preceding discussion demonstrates the need for a comprehensive network failure prediction system. Most relevant research focuses on fiber cut location prediction, failure prediction with domain-specific log data, hard-drive failure prediction with health status, and network failure prediction by predicting congestion. There needs to be a failure prediction system that is compatible with all platforms and combats common failure patterns (described in Chapter 2). Nonetheless, this absence of a network failure identification and prediction system is primarily attributable to poor public failure records. In order to address this research gap, this thesis attempts to compile a network failure dataset with simulated traffic flow. To predict failures, both deep learning and machine learning techniques are applied to the

generated dataset. Our proposed method will serve as a benchmark for future research on the prediction of generic network failure.

Chapter 4

4 Proposed Framework and Methodologies

In this Chapter, the proposed system architecture and experimental setup are detailed. Section 4.1 gives the overall system architecture for data generation and training with ML algorithms. In Section 4.2, the details of the dataset generation process are described. Section 4.3 discusses the experimental setup for the AI techniques used for this study.

4.1 System Architecture

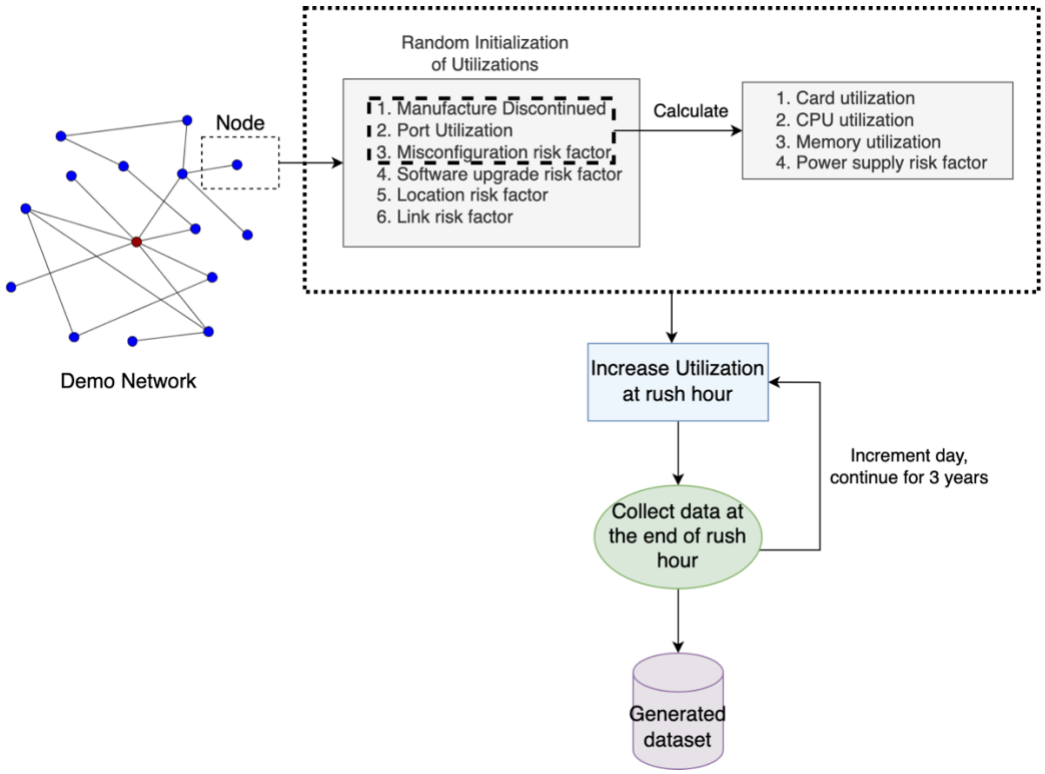

Figure 4. 1: Dataset Generation Overview

Two segments comprise the proposed system: the generation of datasets and the application of Machine Learning and Deep Learning to the generated data for failure prediction. *Figure 4.1* provides an overview of the process of creating datasets. As we can see, we are initially considering a demo network. Every node in this network is a single object with the following attributes: discontinuation of production, port utilization, misconfiguration risk factors, software upgrade risk factors, location risk factors, and link risk factors. We randomly

initialize these variables according to some predefined rules (detailed explanation in 4.2). Manufacture discontinuation, port utilization, and the misconfiguration risk factor are subsequently used to calculate additional properties, including card, CPU, memory, and power supply risk factors [2]. The network traffic simulation begins after initialization, and we collect data at the end of rush hour (6 PM) every day for three years. During rush hour, a random factor taken from a uniform distribution increases the utilization rate. During this time, a single traffic sample from a node is evaluated based on its associated risk factors for different kinds of failure (CPU, memory, Card, Port, Power Supply, Natural disaster failure associated with location, OS upgrades, Misconfiguration, and Link failures). A particular node is selected as a failure category for having a high-risk factor for a specific category or non-failure category. At the end of the simulation, we now have a record of each node's properties and the type of failure (or non-failure) it exhibits based on its properties.

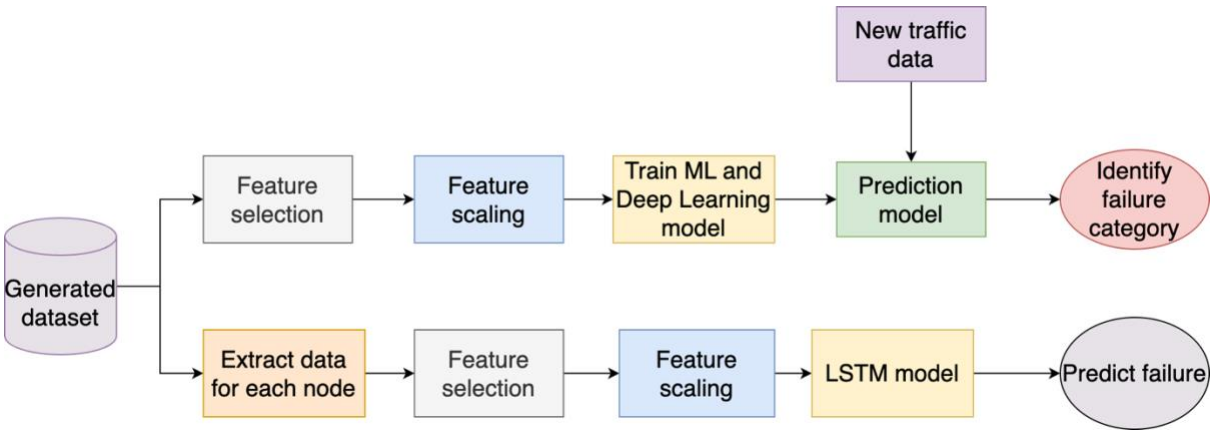

Figure 4. 2: Failure Prediction System with the Generated Dataset

After generating the dataset, we will train the ML model to identify and predict failure (*Figure 4.2)*. The dataset is used in two ways: to identify the failure category of a given new traffic sample and to analyze the sequence of failure statuses of a specific node to predict future failures.

1. **Identify failure category**:

   Most of the features of the generated datasets are considered to build this system. The features are first scaled to have values within a range, making it easier for the ML to converge to a solution. Later this scaled data is fed into several ML and DL algorithms for learning the patterns to identify correct category of failure. After training,

the model is tested with unseen data with new traffic to evaluate its effectiveness in identifying failure categories.

2. **failure prediction:**

   First, the data for each node over three years are separated. We are only considering the failure category as a feature. A portion of this time series data for a single node is scaled to have values between 0 and 1 before being fed into an LSTM model. After training the model, predicted results are evaluated using the remaining data for the node.

4.2 Dataset Generation

In our proposed system, traffic data is simulated for three sample network topologies of varying sizes. We have adhered to Cisco's network failure guidelines to generate the traffic flow [2]. In this section, the entire procedure for generating sample networks and the characteristics of nodes is outlined. The statistical properties of each dataset are subsequently described.

4.2.2 Network architectures

Three different-sized sample network architectures are considered to review the failure data generated by a network thoroughly. The smallest architecture consists of 100 nodes, the medium one contains 200 nodes, and the largest includes 500 nodes. For each network, the minimum nodal degree is three, and the maximum is 10. The total link count for the small network is 638, the medium is 1267, and the largest is 3188. The time to complete the entire simulation process for these networks is 88.7143s, 180.3846s, and 471.1655s, respectively. *Figure 4.3* shows the schematic diagram of the sample network architectures, and *Table 4.1* summarizes the properties of all these networks.

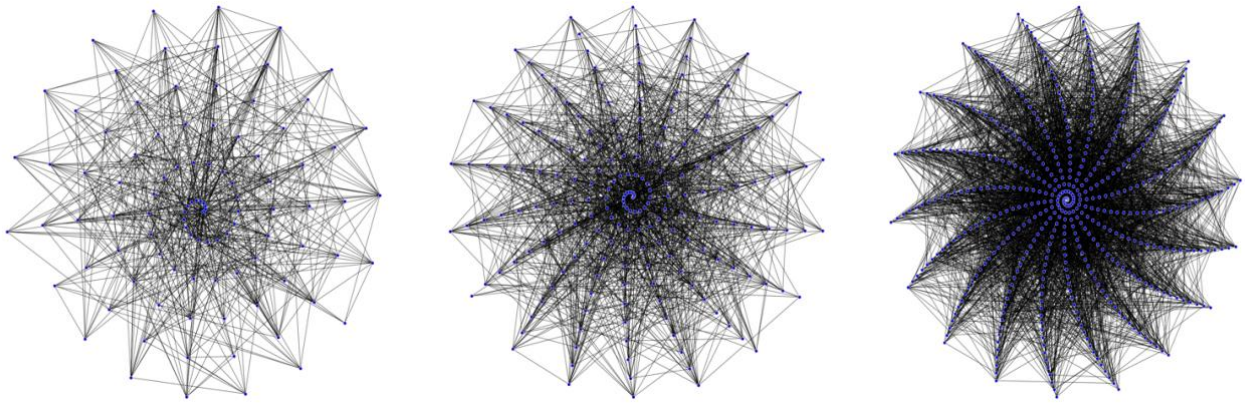

Figure 4. 3: Form the left- Network architecture with 100 nodes, 200 nodes, and 500 nodes.

Each port of every node is randomly initialized with a utilization rate of minimum 0 Gbps and maximum 100 Gbps each port. Uniform distribution is used for this initialization. This port utilization and the adjacency list of the network are used to create a single node.

Table 4. 1: Properties of Sample network architectures

Properties	Network- Small	Network- Medium	Network- Large
Number of nodes	100	200	500
Nodal degree per node	3-10	3-10	3-10
Total number of links	638	1267	3188
Number of cards per node	16	16	16
Number of ports per card	8	8	8
Maximum utilization per port	100 Gbps	100 Gbps	100 Gbps
Memory buffer	10 Gb	10 Gb	10 Gb
Time to generate data	88.7143s	180.3846s	471.1655s
Memory to store data	39.6 MB	79.3 MB	198MB
Total equipment counts	15338	30667	76688

Furthermore, for our simulation of network traffic, following the Cisco guidelines [2], an availability rate of 0.99 is considered for all the sample networks. Another measure of availability, Defects Per Million (DPM), is calculated by multiplying unavailability by 1,000,000. So, for the availability of 0.99, DPM is *1000000 * (1 - 0.99) = 10,000*. This calculation is equivalent to a 1% failure rate of the equipment (10,000 defects per 1000000). Therefore, 1% of the total pieces of equipment can fail at a time for all three sample networks.

4.2.3 Simulation

This section details the overall network architecture creation and traffic simulation processes. Following section provides a description of the properties of each node and the calculation of these properties, the method for generating traffic flow is outlined.

4.2.3.1 Node Properties

To start generating the traffic, Node objects are first created with randomly initialized port utilizations and the adjacency list of the entire network. A single node object has the following properties:

1) ***Name***: The node's name is a numerical value between 0 to 99 or 199, or 499, depending on the network sizes.
2) ***Port capacity***: The maximum port capacity for all eight ports of every card is set to 100 Gbps.
3) ***Card capacity***: Card capacity is *(8 * Port capacity),* with eight ports in one card. There are a total of 16 cards.
4) ***Memory buffer***: The memory buffer is set to 10 Gb.
5) ***Manufacture discontinued:*** This value indicates how long the node is being used. For our case, the maximum time a node can stay operational without much complication is five years (60 months). So, this number is randomly initialized with a uniform distribution with a value between 0 to 60 for all nodes.
6) ***Power supply risk factor***: Referring to *Table 2.1*, manufacture discontinued is the only factor contributing to power supply failure. Therefore, a risk factor in the range of 0 to 10 is assigned to this field using the manufacture discontinued value of the node. The calculation is done as follows:

$$Power\ supply\ risk\ factor = \text{round}(\frac{\text{Manufactured discontinued}}{60} * 10) \qquad \textit{Equation 4.1}$$

7) ***Port utilizations:*** Each port's utilization is randomly initialized with 0 to 100 Gbps with values from a uniform distribution.

8) ***Port risk factor***: *Table 2.1* shows that port failure occurs mainly for a very high utilization rate and secondarily for manufacture discontinued. Accordingly, each port is assigned a risk factor of 0 to 10 based on the following equation:

$$Port\ risk\ factor = \text{round}\left(\frac{\text{Port Utilization}}{10}\right) * 0.9 + \text{round}\left(\frac{\text{Manufactured discontinued}}{60} * 10\right) * 0.1 \quad \textit{Equation 4.2}$$

9) ***Card utilization***: This value indicates the percentage of the card currently being utilized. The sum of all port utilizations for that card is divided by the total card capacity, yielding a number between 0 to 1.

$$Card\ utilization = \text{round}\left(\frac{\text{Sum (Port utilizations)}}{\text{Card capacity}}\right) \quad \textit{Equation 4.3}$$

10) ***Card risk factor:*** Card failure mostly happens due to high utilization and manufacture discontinued (*Table 2.1*). So, each card is assigned a risk factor based on 90% weight on card utilization and 10% on manufacture discontinued.

$$Card\ risk\ factor = \text{round}(Card\ utilization * 10) * 0.9 + \text{round}\left(\frac{\text{Manufactured discontinued}}{60} * 10\right) * 0.1 \quad \textit{Equation 4.4}$$

11) ***CPU utilization:*** CPU utilization depends on the total utilization of each port of the node. The percentage of the CPU in use is determined with the following equation:

$$CPU\ utilization = \text{round}\left(\frac{\text{Sum (Port utilizations)}}{\text{Node capacity}}\right) \quad \textit{Equation 4.5}$$

12) ***CPU risk factor***: The CPU risk factor is between 0 to 10, derived from CPU utilization and discontinued manufacture.

$$CPU\ risk\ factor = \text{round}(\text{ CPU utilization} * 10) * 0.9 + \text{round}\left(\frac{\text{Manufactured discontinued}}{60} * 10\right) * 0.1 \quad \textit{Equation 4.6}$$

13) ***Memory utilization***: Memory utilization of a node depends on the CPU utilization as read/write operation are done during information processing by CPU. But, for memory, there is often a layer of buffer added to hold additional packets to avoid congestion. Memory utilization is calculated using *Equation 4.7*. Here, 100% of the buffered

memory is added with the CPU utilization. Therefore, if we have a CPU utilization of 40%, memory utilization would be 140% with respect to CPU utilization because of the added buffer.

$$Memory\ utilization = \text{CPU utilization} + \frac{\text{Memory buffer}}{10} \qquad \textit{Equation 4.7}$$

14) ***Memory risk factor***: This is a value between 0 to 10, indicating the risk of memory failure. It primarily relies on the very high utilization of memory components and secondly on manufacture discontinued (*Table 2.1*).

$$Memory\ risk\ factor = \text{round}(\text{Memory utilization} * 10) * 0.9 + \text{round}\left(\frac{\text{Manufactured discontinued}}{60} * 10\right) * 0.1 \qquad \textit{Equation 4.8}$$

15) ***Link failure risk factor***: With the adjacency list of the network, each link connecting to other nodes is assigned a random risk factor between 0 to 10 for fiber cut, the primary cause of link failure. The secondary cause of link failure is a configuration error. First, the risk factor for each link is calculated with 90% weight on the fiber-cut risk factor and 10% weight on the misconfiguration risk factor. Later, the average of all outgoing links is taken as the final value.

$$Link\ risk\ factor = \frac{\text{Sum}((\text{fiber cut risk factor} * 0.9) + (\text{misconfiguration risk factor} * 0.1))}{\text{Total link count}} \qquad \textit{Equation 4.9}$$

16) ***Location risk factor***: This risk factor is associated with natural disasters and other causes. This factor is assigned with a random number between 0 to 10.

17) ***Misconfiguration risk factor***: The chances of a misconfiguration will depend on the history of this kind of error on that specific device. This node property is given a random risk factor between 0 to 10.

18) ***OS upgrade risk factor***: Chances of failure due to OS upgrades are common (*Table 2.1*). A random integer value between 0 and 10 from a uniform distribution is assigned for each node indicating the risk factor associated with the OS upgrade.

Overall, each node has a manufacture discontinued value ranging from 0 to 60, ports utilization of a maximum of 100 Gbps, memory buffer of 10 GB. Additionally, the risk

factor ranges from 0 to 10 for the power supply, port, card, CPU, memory, link, location, misconfiguration, and OS upgrade.

***Algorithm 1*:** Traffic generation

Input: nodes with manufacture discontinued, port utilizations and risk factors, final dataset = {}, month counter = 0

Output: final dataset = traffic data samples of a category- no failure, CPU, memory, card, port power supply, location, OS upgrade, misconfiguration, link failure

```
For (each quarter of three years' time)
    If (one month passed)
        Month counter = 1
    Else
        Month counter = 0
    Foreach node do
        If (month counter == 1)
            Manufacture discontinued++
        If (rush hour)
            Increase port utilization up to 40%, not crossing 100 Gbps
            Re-calculate risk factors
            Add data sample to the final dataset:
            If (risk factor >= 5)
                final dataset += Failure category
            Else
                final dataset += Normal traffic
        Else
            Keep the original risk factor and utilization rate
    End for
End for
Return final dataset
```

4.2.3.2 Traffic generation

After establishing the nodes for small, medium, and large networks, three years' worth of traffic is generated. Each day is divided into four quarters for this purpose: 12 AM to 6 AM, 6 AM to 12 PM, 12 PM to 6 PM, and 6 PM to 12 AM. While maintaining a maximum speed of 100 Gbps, port utilization is increased by a random factor taken from a uniform distribution of up to 40% during rush hour. All node properties are recalculated in response to an increase in port utilization. At the end of each month, *manufacture discontinued* is increased by 1. For data collection after each rush hour, the nodes with the highest risk factors in a particular category are selected as candidates for that failure category. Otherwise, it is designated as a standard traffic flow. Equipment with a risk factor of 5 or higher is considered high-risk. For instance, a node with a misconfiguration risk factor of nine is selected as the category for misconfiguration failure. The entire procedure is depicted in Algorithm 1.

As mentioned in the CISCO guidelines section, for all the sample network topologies, a 0.99 availability rate is considered, i.e., 1% defects (10,000 DPM). Among these 1% defects, the ratio of each type of failure is maintained based on *Table 2.1*. Finally, three different datasets are created for the three different topologies.

4.2.4 Statistical properties of dataset

In this subsection, the statistical properties of the dataset are described. Each row of the dataset for three different topologies contains the following features:

1) **Date**: DateTime object
2) **Name**: number of the node
3) **Manufacture discontinued**: Integer value between 0-60
4) **Power supply risk factor**: Integer value between 0-10
5) **CPU risk factor**: Integer value between 0-10
6) **Memory risk factor**: Integer value between 0-10
7) **Location risk factor**: Integer value between 0-10
8) **Misconfiguration risk factor**: Integer value between 0-10
9) **Average link risk factor**: Integer value between 0-10 (ceiling of the average)
10) **Card 0 to Card 15 risk factor**: Integer value between 0-10 for each card

11) **Port 0 to port 7 risk factor for each card**: Integer value between 0-10 for each port
12) **Failure Category- Target**: Integer value between 0-9 where
    - 0 - No failure
    - 1 - CPU
    - 2 - memory
    - 3 - Card
    - 4 - Port
    - 5 - Power Supply
    - 6 - Location
    - 7 - OS upgrade
    - 8 - Misconfiguration
    - 9 - Link
13) **The number of data points, and memory usage per data frame**:
    - Small network (100 nodes): 109500, 136.6 MB
    - Medium network (200 nodes): 219000, 273.2 MB
    - Large network (500 nodes): 547500, 683.0 MB

Correlation graphs, bar plots, and violin plots are analyzed for each dataset to get an overall sense. The following fields are considered for this: manufactured discontinued, power supply risk factor, CPU risk factor, memory risk factor, location risk factor, misconfiguration risk factor, OS risk factor, average link risk factor, an average of all port risk factors, an average of all card risk factors, and failure categories.

At first, from the correlation graph for all three networks (*Figure 4.4, Figure 4.6, and Figure 4.7*), we can see that the target attribute (failure category) is correlated to some degree with all the other features. Significantly, the most strongly correlated feature with the target is the link failure risk factor for all the networks. Other significant ones are misconfiguration, OS upgrades, and card and port risk factors. All other risk factors have a mild correlation with the dataset's target. Therefore, we can see that the generated datasets follow the failure percentages mentioned in *Table 2.1*.

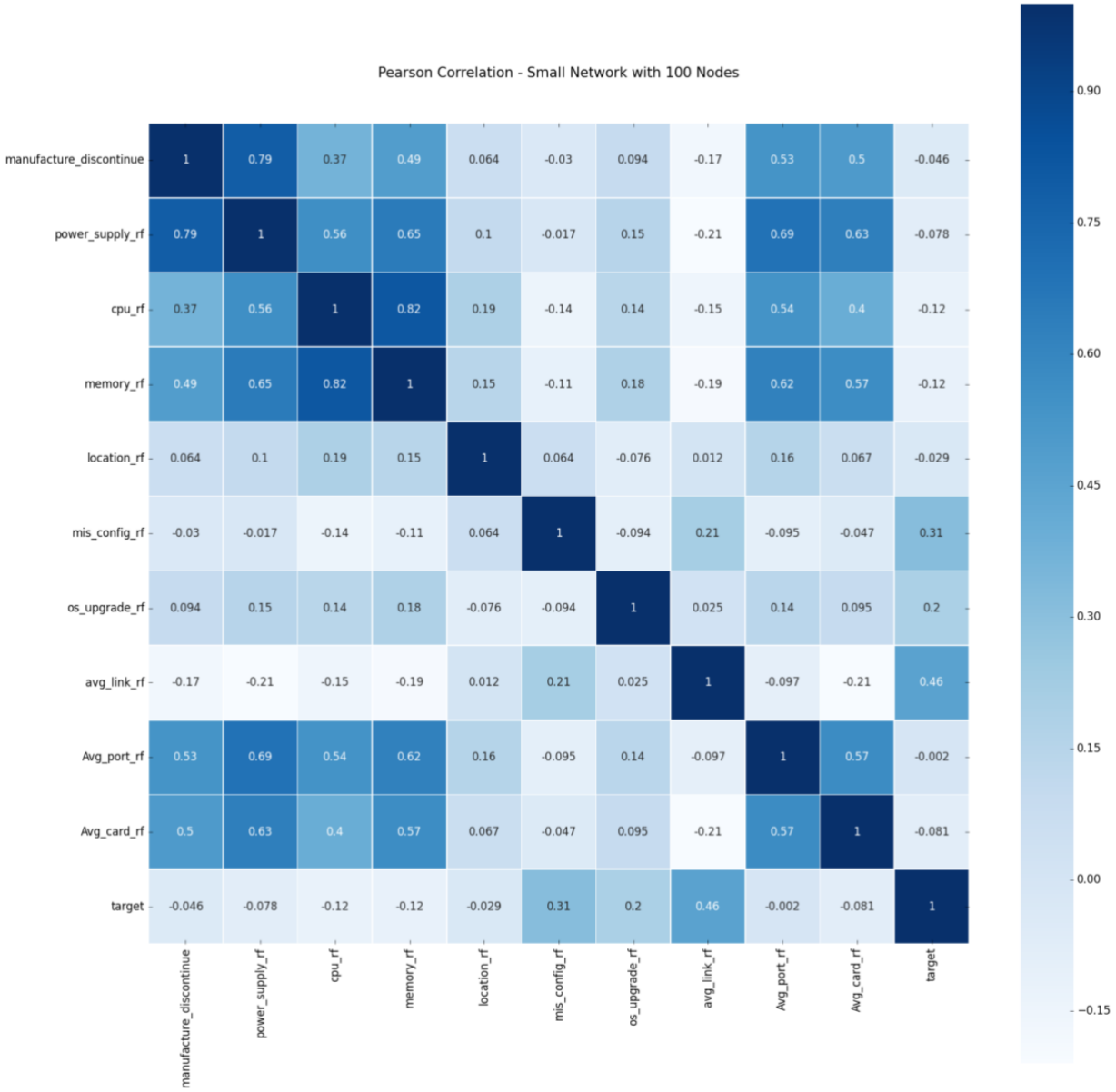

Figure 4. 4: Pearson Correlation Map - Small Network (100 Nodes)

Figure 4. 5: Pearson Correlation Map - Medium Network (200 Nodes)

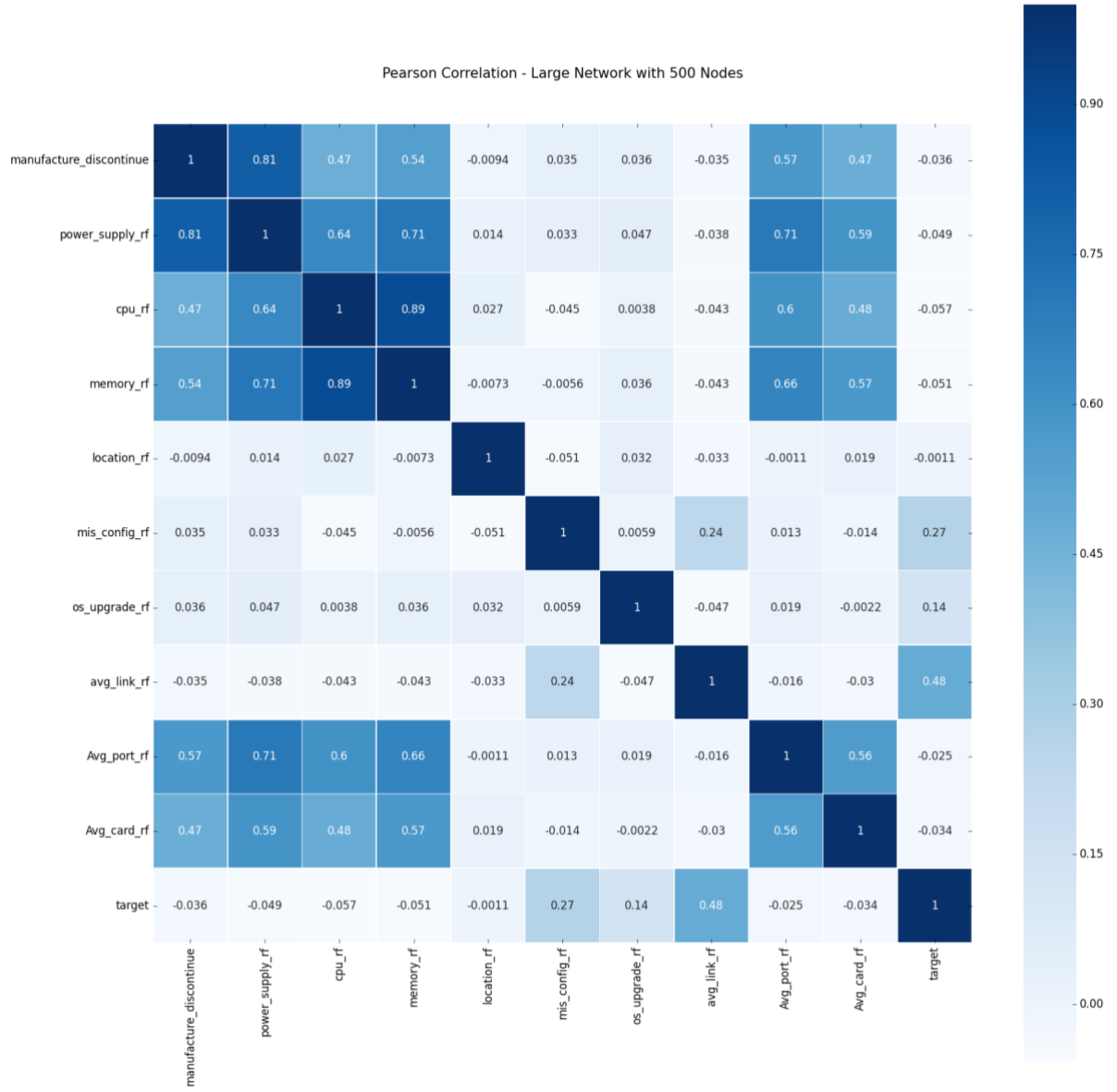

Figure 4. 6: Pearson Correlation Map - Large Network (500 Nodes)

Figures 4.7- 4.18 illustrate bar and violin plots for all three dataset's properties. As can be seen, most devices have a discontinuation rate greater than 50 across all three datasets. This is because we have collected traffic data for three years. The violin plots illustrate the overall distribution of each property. Again, due to random initialization, the power supply, location, misconfiguration, and operating system risk factors follow a uniform distribution. The risk factor associated with the power supply is directly attributable to the manufacturer's discontinuation.

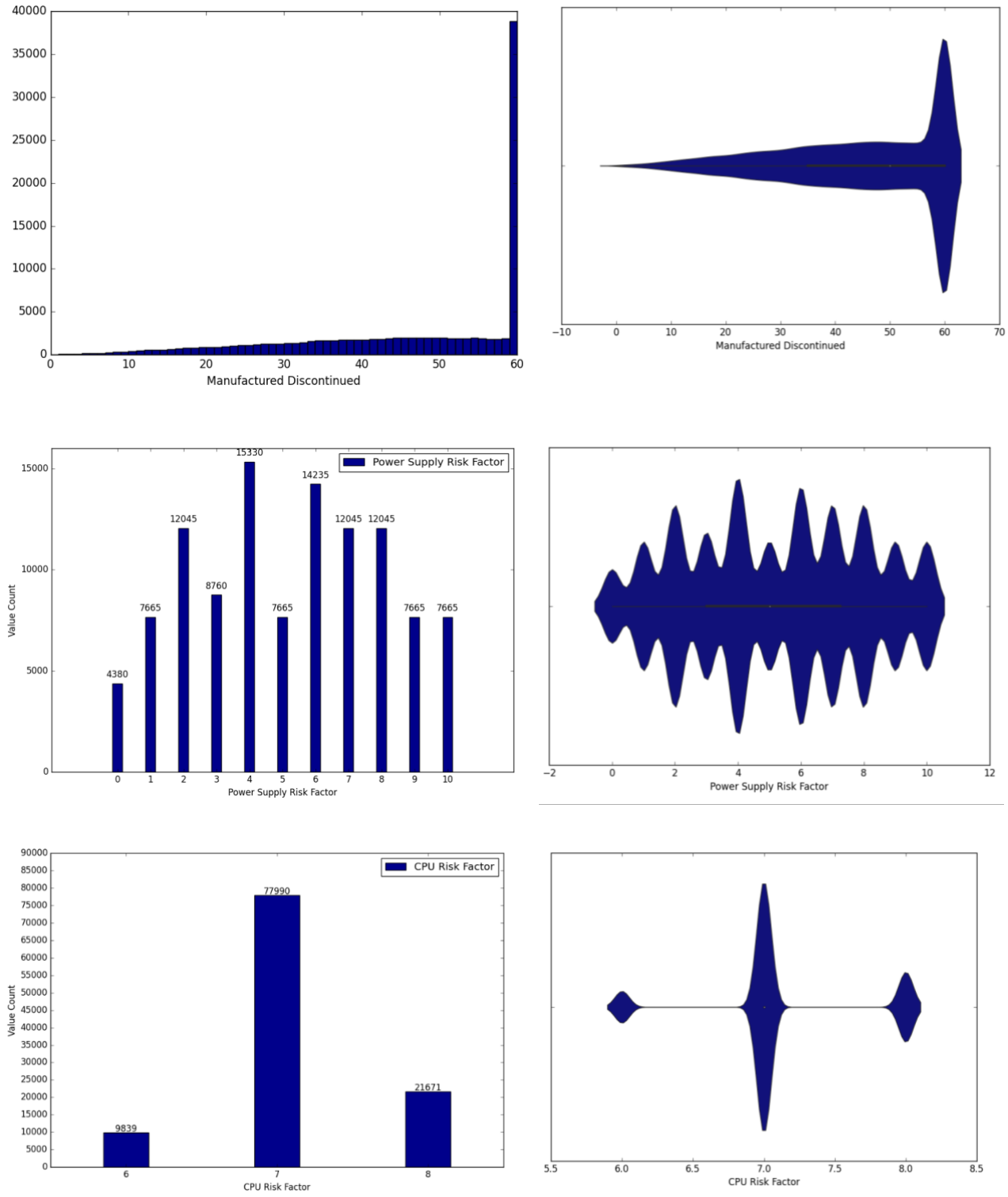

Figure 4. 7: Left- Bar Plots, Right- Violin Plots of features of the Small Network (100 Nodes)

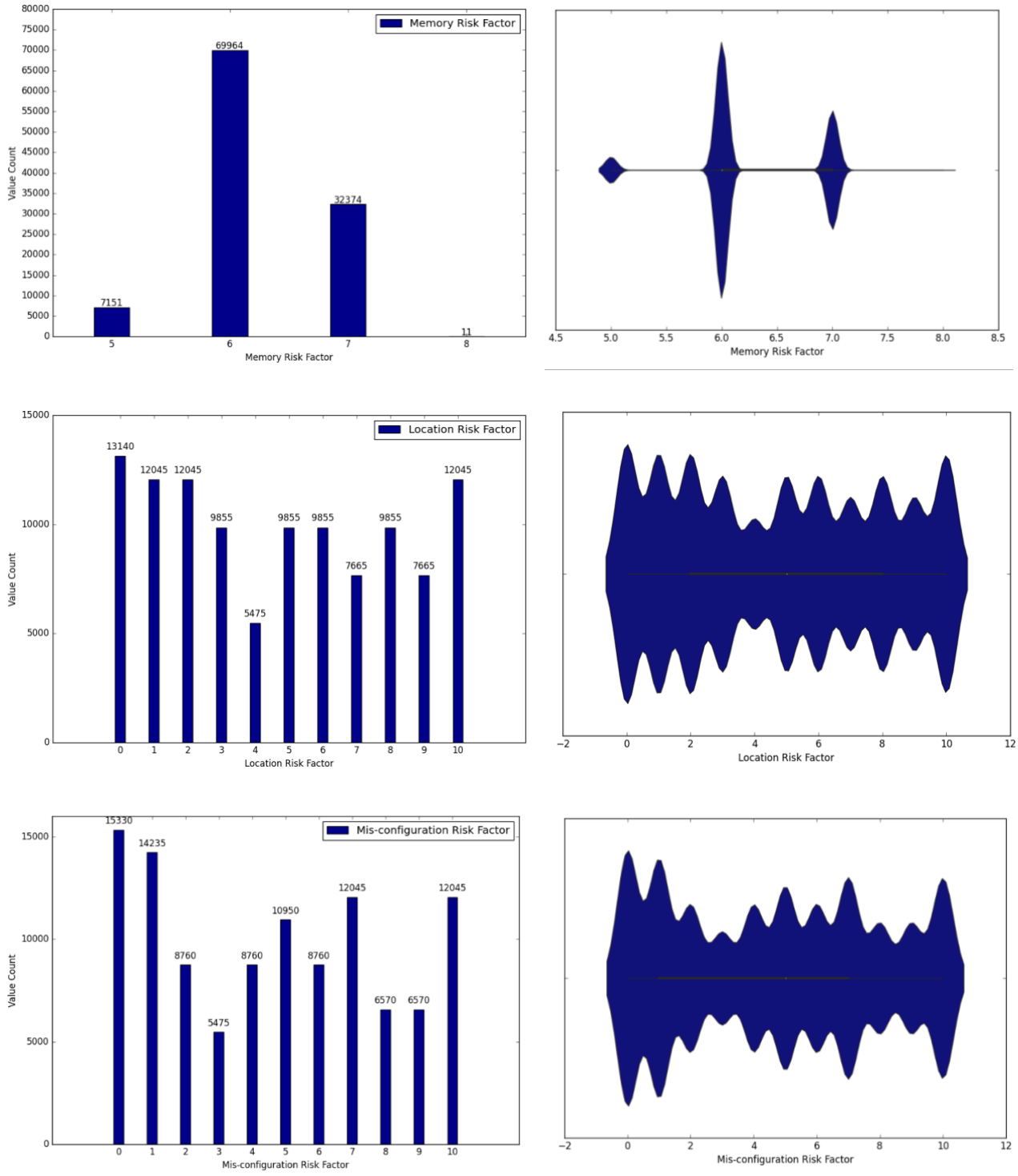

Figure 4. 8: Left- Bar Plots, Right- Violin Plots of features of the Small Network (100 Nodes)

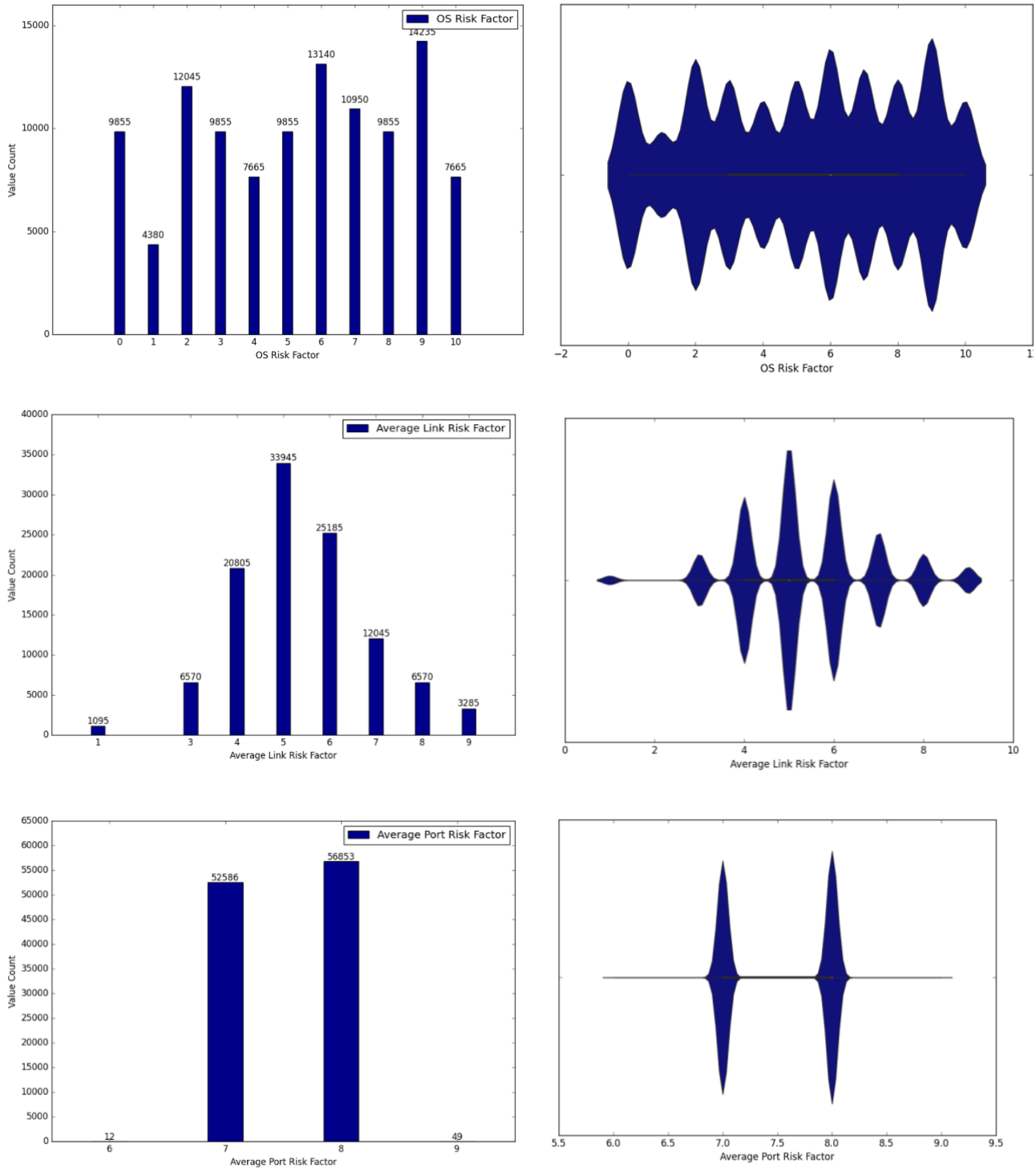

Figure 4. 9: Left- Bar Plots, Right- Violin Plots of features of the Small Network (100 Nodes)

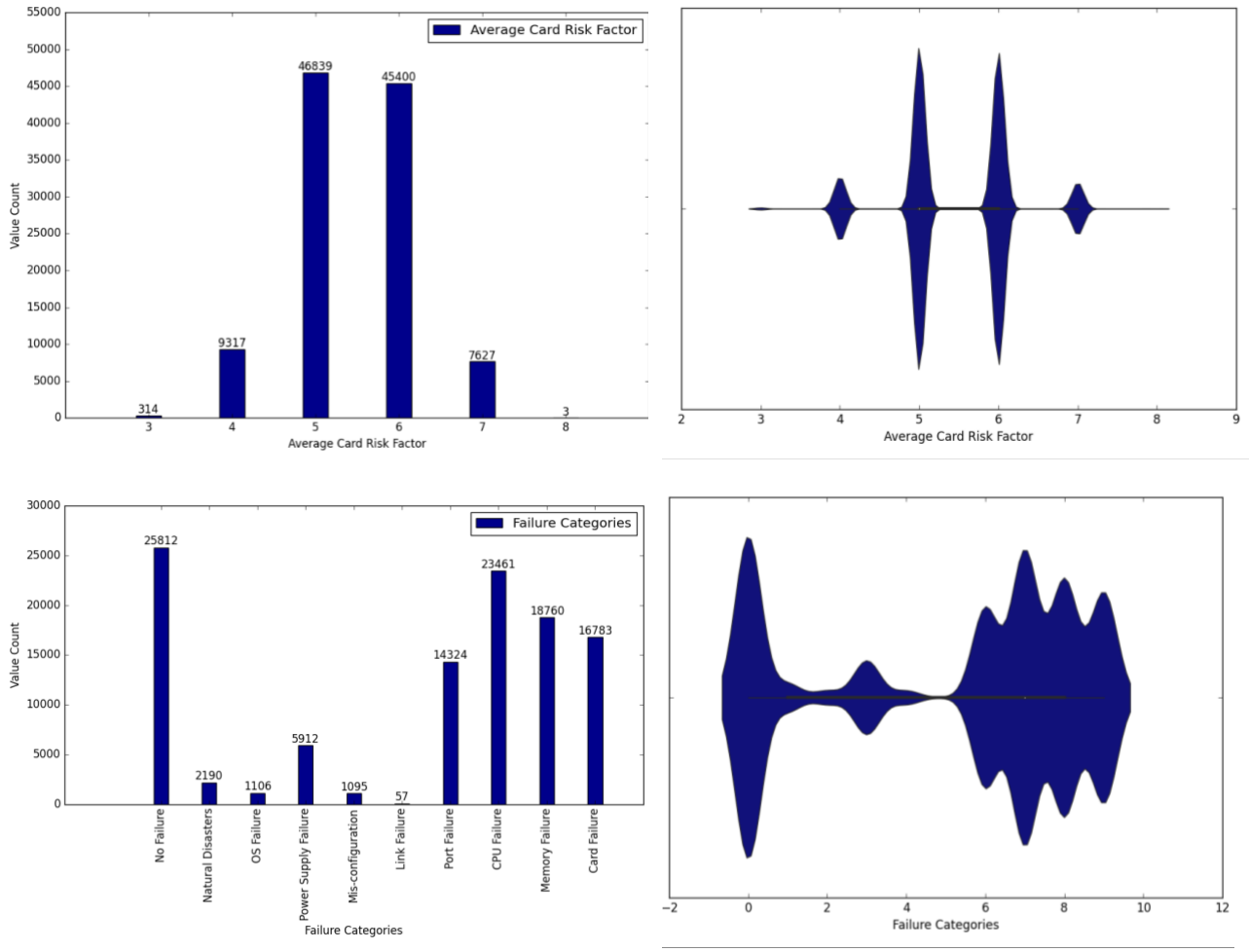

Figure 4. 10: Left- Bar Plots, Right- Violin Plots of features of the Small Network (100 Nodes)

Furthermore, the other risk factors, including CPU, memory, average link risk factor, average port risk factor, and average card risk factor, follow a normal distribution for all three datasets. All these risk factors are primarily derived from the port utilization rate, which is randomly generated with uniform distribution (Node Properties subsection, Page 35). All these risk factors are derived mainly from the average port utilization. Moreover, according to the Central Limit Theorem, the average of sizeable uniform distribution tends toward the normal distribution. Additionally, we can confirm a strong correlation between all these properties from the Pearson correlation graphs of *Figures 4.4 - 4.6.*

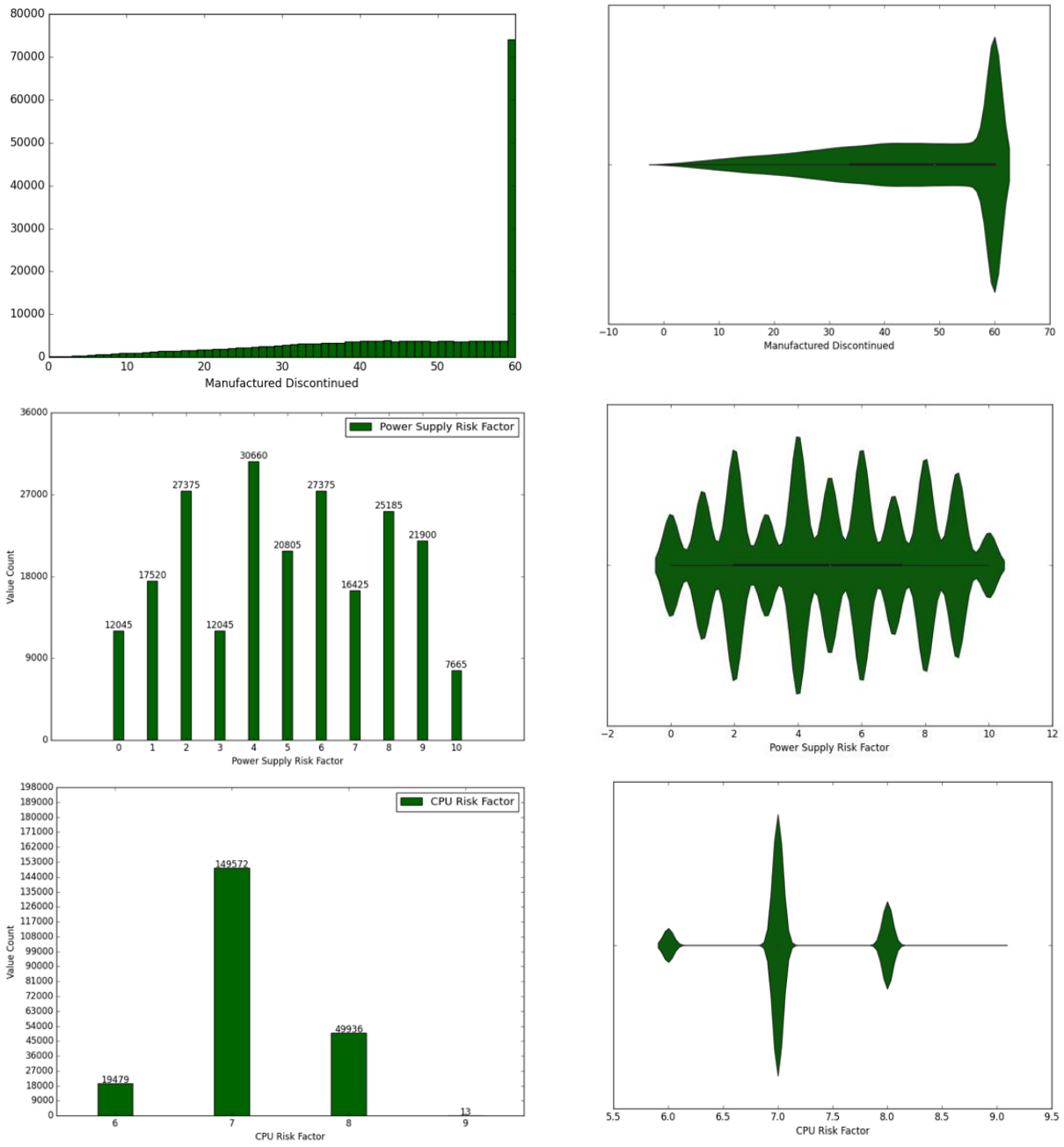

Figure 4. 11: Left- Bar Plots, Right- Violin Plots of features of the Medium Network (200 Nodes)

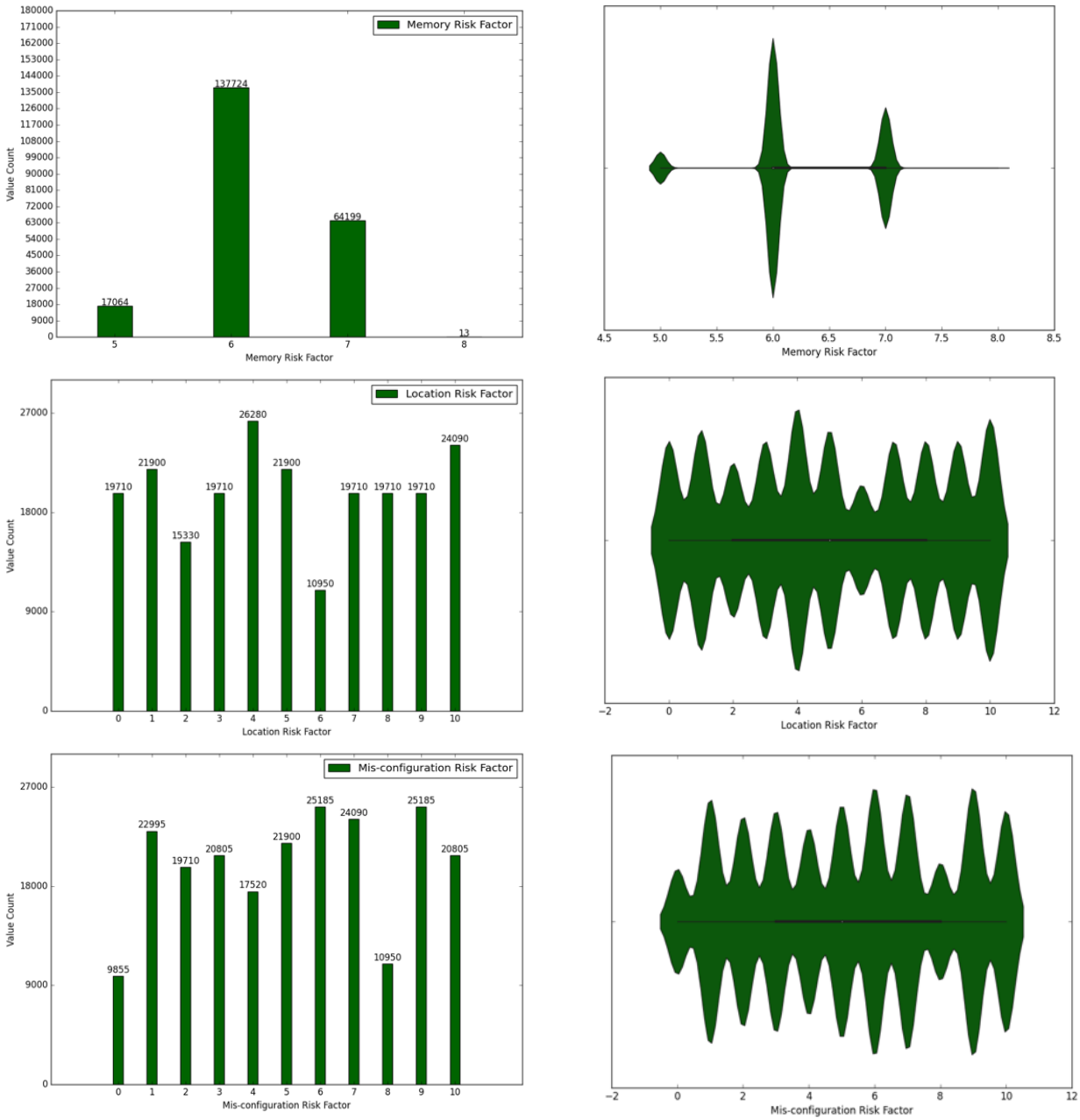

Figure 4. 12: Left- Bar Plots, Right- Violin Plots of features of the Medium Network (200 Nodes)

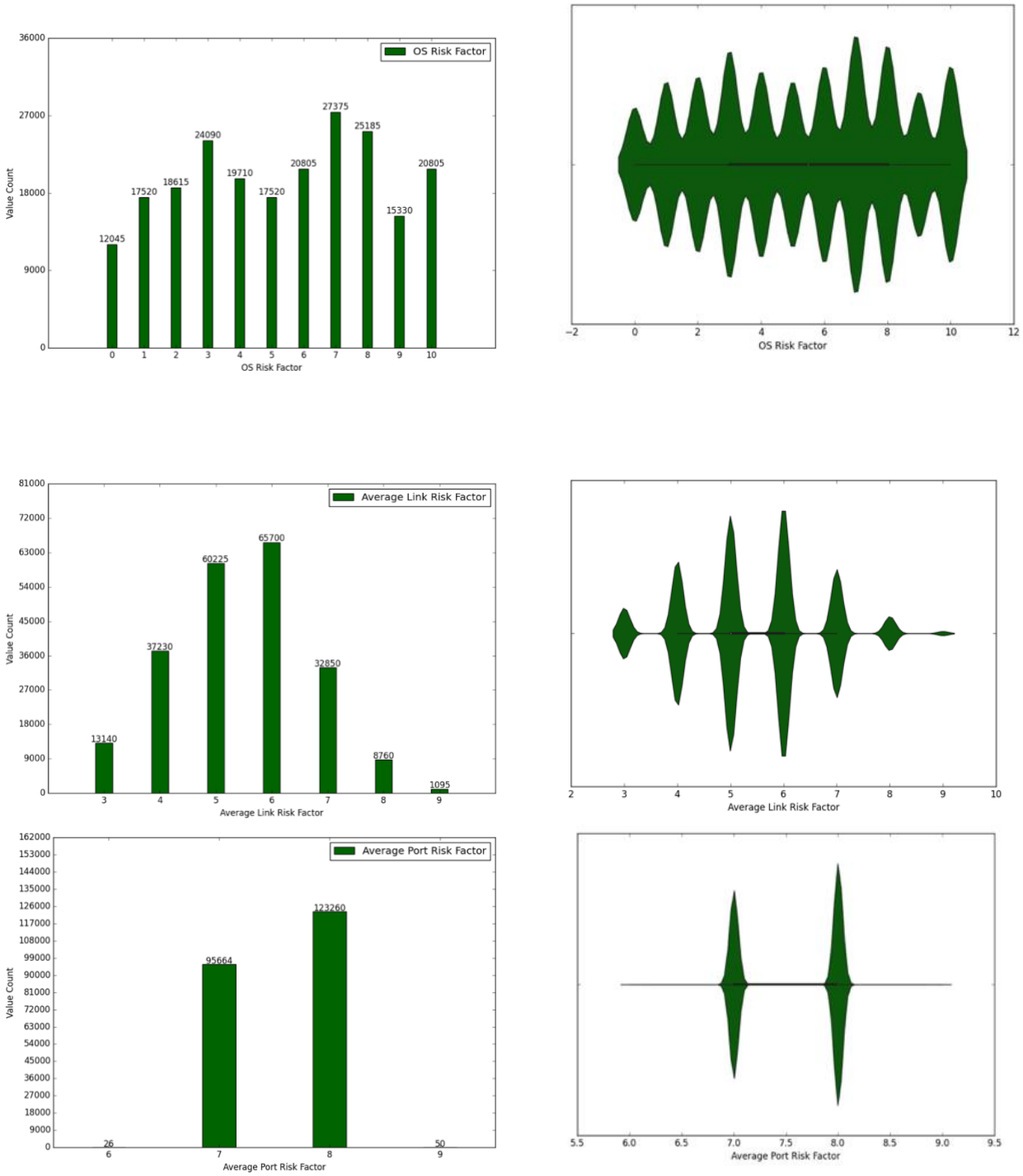

Figure 4. 13: Left- Bar Plots, Right- Violin Plots of features of the Medium Network (200 Nodes)

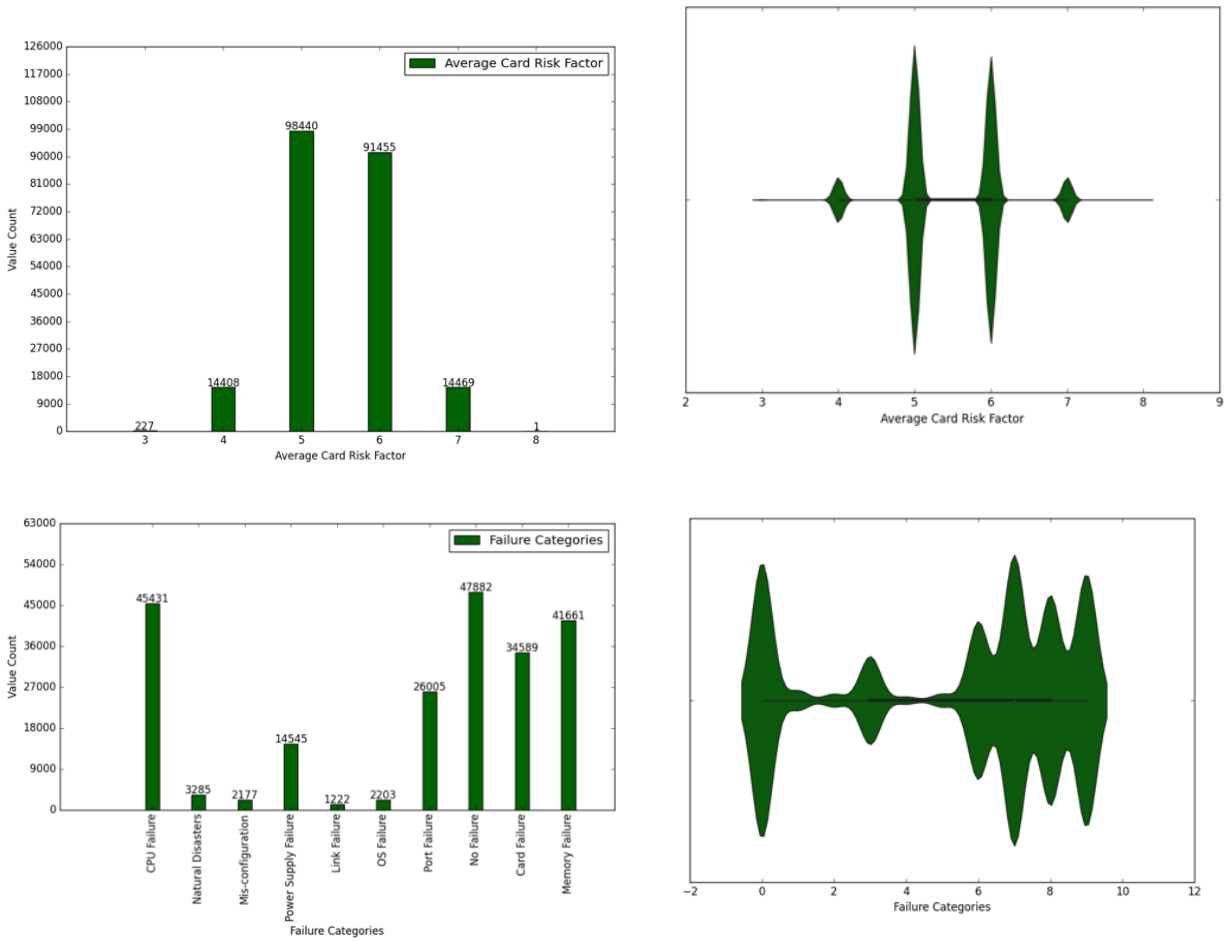

Figure 4. 14: Left- Bar Plots, Right- Violin Plots of features of the Medium Network (200 Nodes)

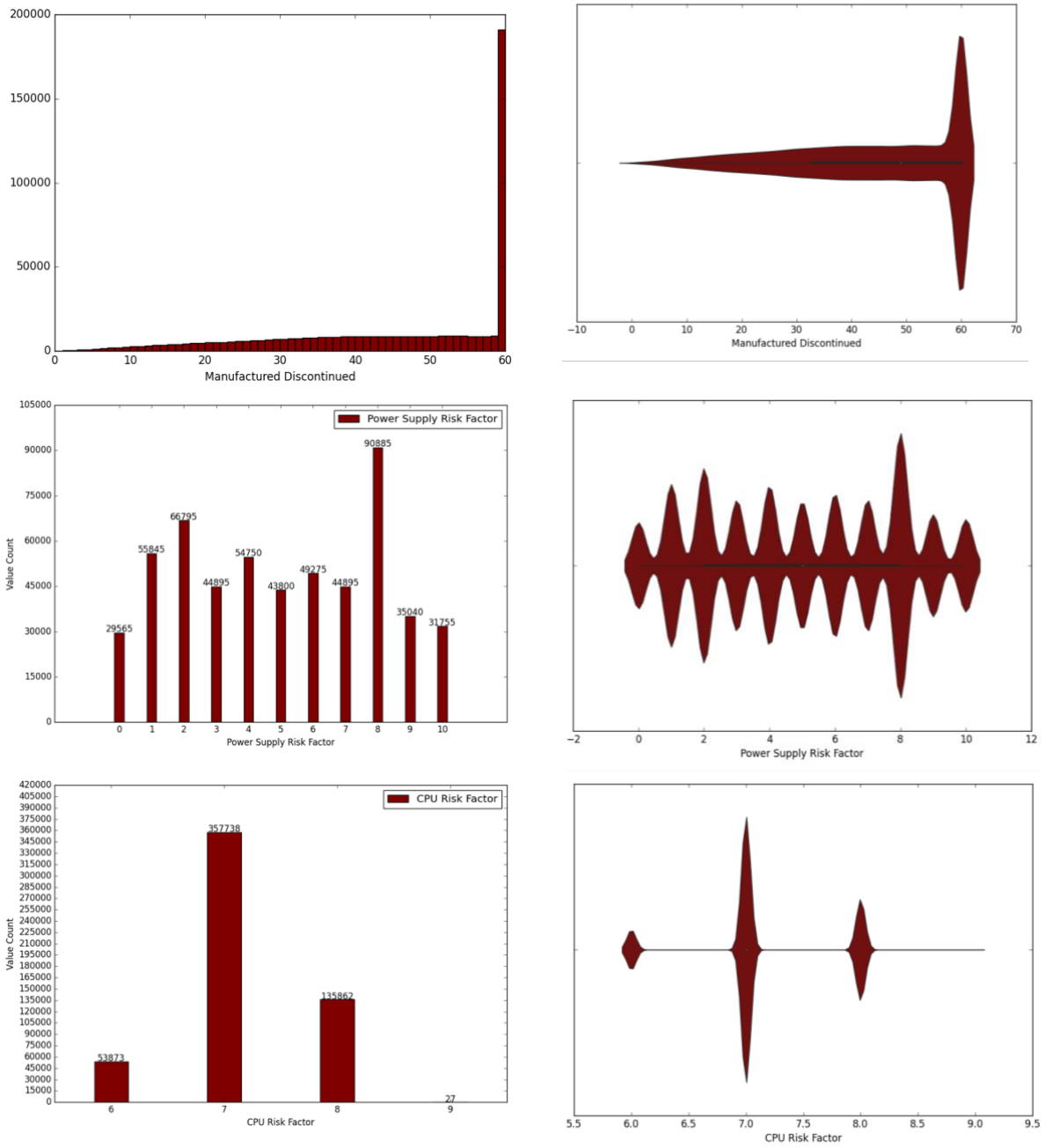

Figure 4. 15: Left- Bar Plots, Right- Violin Plots of features of the Large Network (500 Nodes)

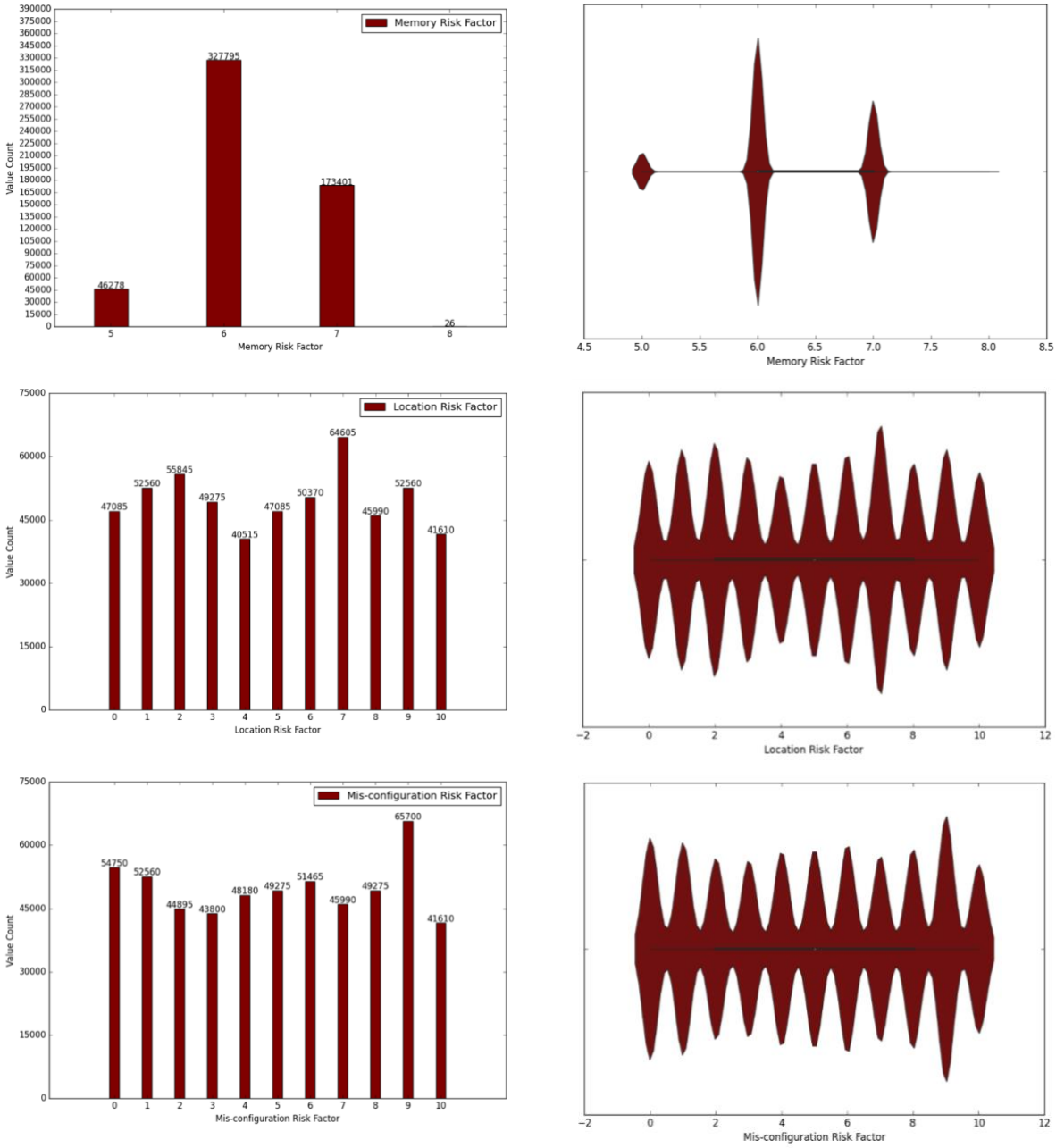

Figure 4. 16: Left- Bar Plots, Right- Violin Plots of features of the Large Network (500 Nodes)

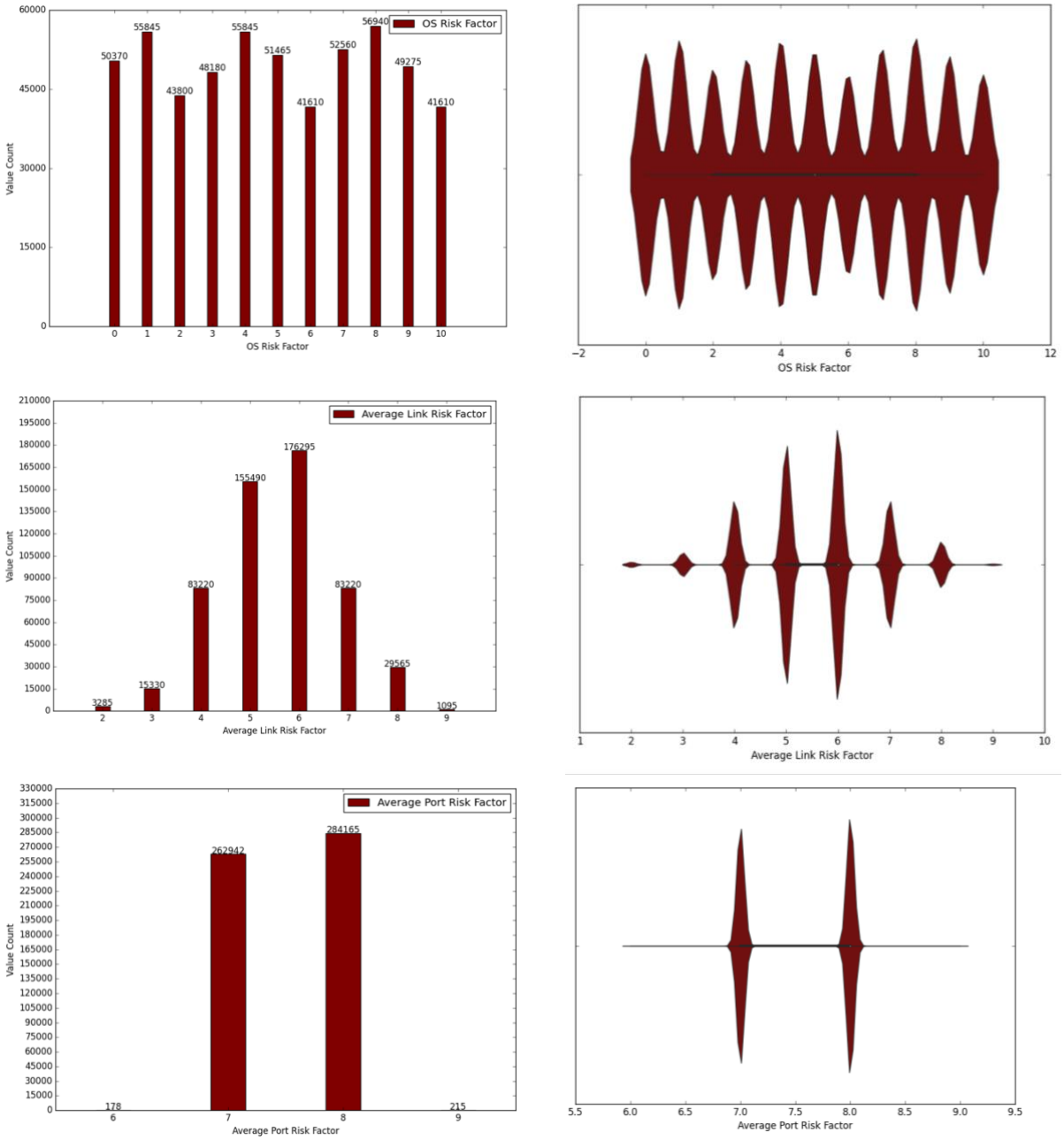

Figure 4. 17: Left- Bar Plots, Right- Violin Plots of features of the Large Network (500 Nodes)

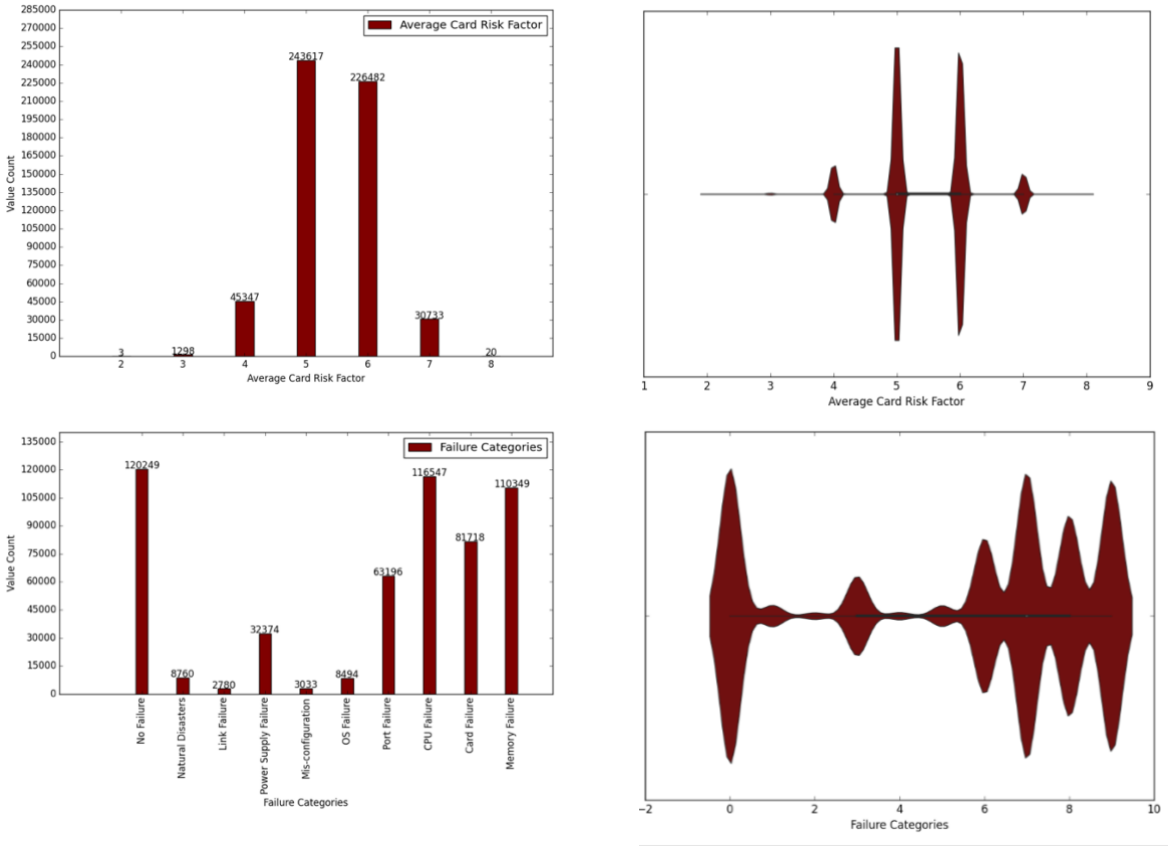

Figure 4. 18: Left- Bar Plots, Right- Violin Plots of features of the Large Network (500 Nodes)

Finally, the bar plots for the failure category class for all three datasets show that the most significant samples are for the *no-failure* category. The other prominent categories are average link, OS, location, and misconfiguration risk factors. The port, card, CPU, memory, and power supply failures have the lowest count. This distribution shows that the failure and non-failure data follow the Cisco guidelines [2]. However, the datasets demonstrate an imbalanced characteristic, as some failure categories have fewer entries than others. Overall, the above discussion demonstrates the soundness of the datasets.

4.3 Training with AI Methods

Several cutting-edge ML and Deep Learning (DL) techniques have been applied to the generated datasets to investigate the best possible option in terms of performance. Different algorithms bring diverse perspectives to the data by capturing unique patterns and relationships among the features. By using a variety of techniques, we reduce the chance of relying on a single algorithm's biases or constraints, resulting in a more thorough comprehension of the data

revealing subtle insights. Therefore, a number of ML and DL techniques commonly followed in recent literature are evaluated on the generated dataset. Training time and memory of these algorithms are also considered for the overall experiment. This section contains the experimental setup and the steps to prepare the dataset for the methods.

4.3.1 Experimental Setup

The experiment is performed on a MacBook Air (2020) with an M1 processor and 8GB of memory. All experiments utilize Python as their primary language. Python's sci-kit-learn library is used for the Machine Learning algorithms. In addition, the Keres framework with a TensorFlow backend is utilized for deep learning. Pandas, NumPy, matplotlib, and seaborn are among the additional data manipulation and visualization modules deployed.

Table 4. 2: Train Test split for each dataset

	Small dataset – 100 nodes	**Medium dataset – 200 nodes**	**Large dataset – 500 nodes**
Training data	87600	175200	438000
Testing data	21900	43800	109500

In order to prepare the data for ML and DL algorithms, it is divided into training and test data. As a training test split 80%-20% ratio is used for all datasets. The number of training and testing data points is shown in *Table 4.2*. As there is an imbalance in the dataset, the Synthetic Minority Oversampling Technique (SMOTE) is applied to the datasets, and the experimental results are compared for a thorough investigation. *Table 4.3* contains the train test split with SMOTE applied to the datasets [72]. All categories contain equal number of data points in this case. SMOTE is applied only to the training data, and the test data is the same as the original set.

Table 4. 3: Train Test split for each dataset – with SMOTE

	Small dataset – 100 nodes	**Medium dataset – 200 nodes**	**Large dataset – 500 nodes**
Training data	206270	383050	961960
Testing data	21900	43800	109500

SMOTE creates new synthetic data samples of the minority categories to balance the dataset. SMOTE work by taking some random minority samples and interpolating the points of the straight line between these samples. For each feature f, if x is a minority class sample and x' is one of its neighbors then, $\Delta = x_j - x'_j$. A new sample would be calculated by: $x + \lambda * \Delta$ where λ is a random number between 0 and 1. The train test splits with SMOTE applied on all three datasets are shown in *Table 4.3*.

4.3.2 Training with ML and DL

After splitting the data into training and testing sets, we selected several features before training them with the ML and DL algorithms. For creating the model that will predict the failure category, all dataset features except the *node identifier* and the *date* are considered. These columns are dropped from the data set as a preparation step. After we select the features for the training, feature scaling is done to normalize the data as a preprocessing step [73]. The training data is first fitted with a standard scaler which follows the following equation:

$$s = \frac{(x - \mu)}{\sigma^2} \qquad \textit{Equation 4.10}$$

Here μ and σ^2 are the mean and standard deviation of the training data. The standard scaler brings all the numbers to the same scale for faster convergence and accurate results. The scaler fitted with training data scales the test data to prevent leaks.

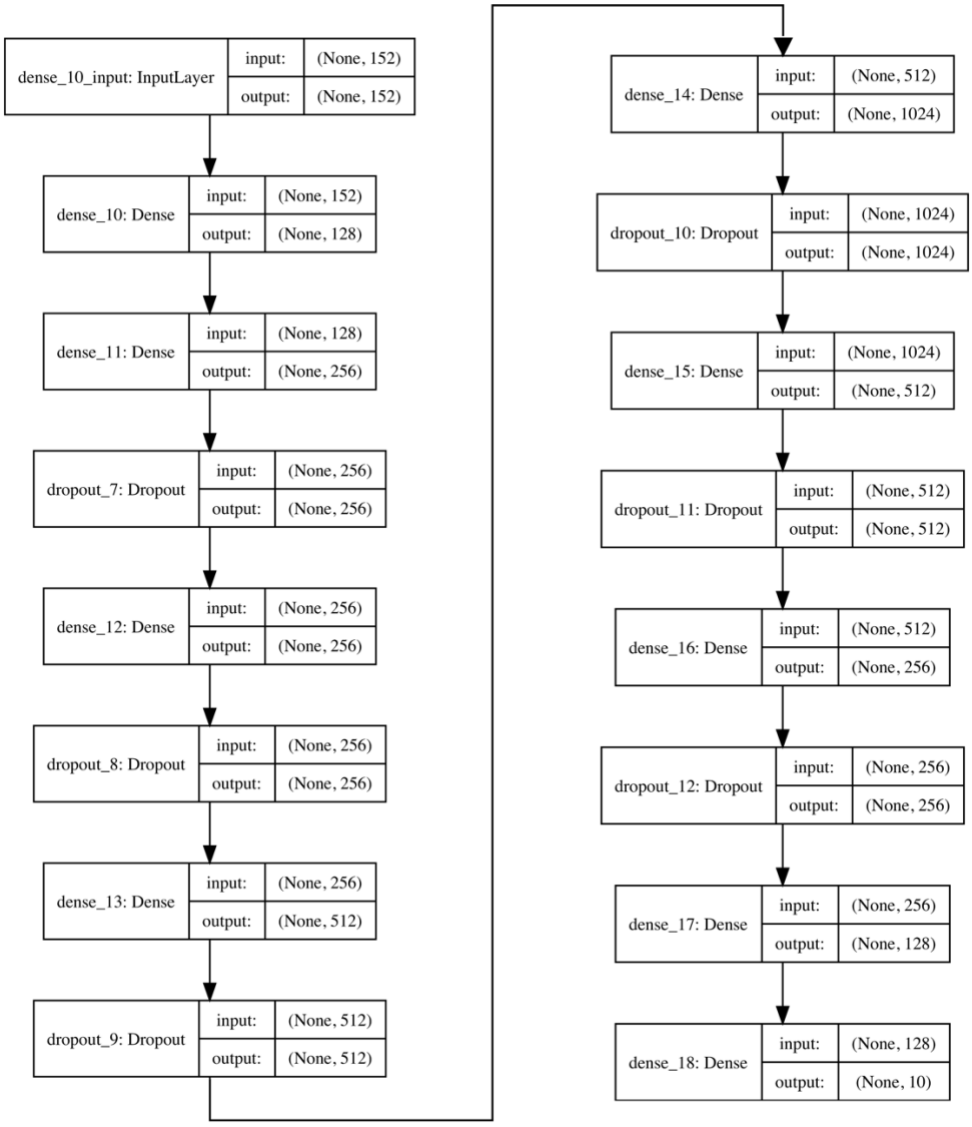

Figure 4. 19: Proposed Deep Neural Network

After these steps, the data is trained with all the ML algorithms described in Chapter 2. The optimal hyperparameters are decided with grid search for all the state-of-the-art ML techniques. For logistic regression, *l2 regularization* and '*saga'* solver are applied to get the best performance. The proposed decision tree model considers *gini impurity* for calculating information gain with a *maximum depth of 10*. Also, the minimum number of samples required to make a split is set to *10*. In the case of a random forest, the number of trees in the forest is set to *50*. *Gini impurity* with a *maximum tree depth* of *10* is used like the decision tree model. But, in the case of a random forest, the minimum number of samples required to make a split

is set to *2*. For SVM, two types of kernels, namely, 'linear' and 'rbf' kernel is considered. The proposed AdaBoost model has a *learning rate of 0.01* and *50* weak classifiers for majority voting. The Gradient boosting algorithm, on the other hand, has *0.1* as the learning rate and *150* weak classifiers for majority voting. For the MLP algorithm, the *'reLu'* activation function with a hidden layer of *100* nodes is considered. For Naive Bayes and QDA, there was no hyperparameter tuning involved as they operate based on probability theory. For training all these models fivefold cross validation is applied.

However, *Figure 4.19* shows our proposed DNN architecture. This model's hyperparameters are fine-tuned with several trials and errors to have the optimal number of *layers*, *learning rate*, and *epochs*. The input layer of the proposed DNN model takes the 152 features of the dataset as input. Then we have 2 hidden layers of the dimensions 128 and 256. A bias term is added for all the hidden layers of the model. Following the two layers, we have a *dropout* layer with a 10% *dropout* rate. *Dropout* layers give a regularization effect in the model by randomly deactivating certain percentages of nodes of that layer. After the *dropout* layer, we have a number of dense-dropout blocks. The rest of the dense layers contain 256, 512, 1024, 512, 256, and 128 blocks consecutively. Here, the *dropout* layers added after these dense layers have a dropout rate of 50%, 50%, 25%, and 10%. At the end, we have a dense layer with 10 nodes which uses the *Softmax* function to give a probability of a particular data sample belonging to a specific class.

Each dense layer uses the *ReLu* activation function to add non-linearity to the data. *ReLu* activation is defined as: $f(x) = max(0, x)$, i.e., the function returns the input value x if it is positive and returns zero otherwise. As a loss function, *sparse categorical cross entropy* is used with *Adam (Adaptive Moment Estimation)* optimizer [74], [75]. The model is trained for 20 epochs with a learning rate of 0.0001.

Overfitting is an important issue when we are considering synthetic data. If we have an intricate model with a number of interdependent features, an ML model can learn to overly fit in the associated mathematical functions. In such a case, our model will perform well on the training data and perform poorly on the test data. Therefore, to avoid such overfitting issues proper regularization is considered for all the proposed ML and DL models to get a generalized solution to the problem. For logistic regression, l2 regularization is used. Random Forest avoids overfitting by taking a number of decision trees and applying the majority voting technique.

The proposed boosting algorithms similarly avoid overfitting by taking majority voting from the outcome of multiple weak classifiers. The DNN model uses multiple dropout layers to combat overfitting.

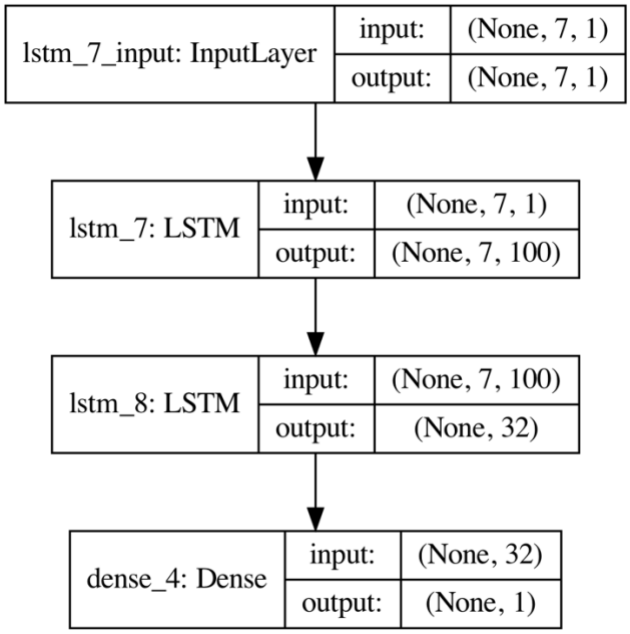

Figure 4. 20: Proposed LSTM Model

4.3.3 Training with LSTM

We have utilized LSTM to predict the availability of a single model node. For this purpose, data samples from each node are isolated sequentially for three years. In this instance, we only utilize the availability information (the *'target'* column). The selected column is then scaled using a min-max scaler for each node [73]. *Equation 4.11* is a representation of the min-max scaler.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad \textit{Equation 4.11}$$

Here, X_{max} is the maximum number and X_{min} is the minimum number of the list X.

After these preprocessing stages, the proposed LSTM model (*Figure 4.20*) is provided with the sequence data. Since we predict one output at a time, the proposed model consists of two LSTM blocks with 100 and 32 nodes, followed by a dense layer with one node. These

layers have the activation function 'ReLu.' Seven days of failure data are regarded as a single input set, with the eighth-day availability status appended as a label. The loss function used for training is the Root Mean Square Error. The model is trained for 10 epochs with a learning rate of 0.0001.

Chapter 5

5 Experimental Evaluation

This Chapter explains the experimental results of every method. The following subsections describe the results of different state-of-the-art machine learning and deep learning techniques with and without SMOTE, followed by the results for the LSTM model.

5.1 Experimental Results, Time-Space: Original datasets

As performance evaluation metrics, *accuracy*, *precision*, *recall*, and *F1-scores* are used for the ML and DL models. *Accuracy* gives us an overall measure of correctly identified data points among all the data points. *Precision* quantifies the quality of the positive prediction with a ratio of correctly predicted positive samples to all predicted *positive samples*. On the other hand, *recall* calculates the rate of correctly predicted positive samples among all correctly predicted *samples*. The *F1-score* represents the harmonic mean of *precision* and *recall* giving an overview of the model's performance. We can calculate these matrices with the numbers of true positives (number of correctly predicted positive samples), true negatives (correctly predicted negative samples), false positives (incorrectly predicted positive samples), and false negatives (incorrectly predicted negative samples). *Equations 5.1–5.4* provide the mathematical formulas for these matrices.

$$Accuracy = \frac{True\ Negative + True\ Positive}{True\ Negative + False\ Positive + True\ Positive + False\ Negative} \quad \textit{Equation 5.1}$$

$$Precision\ = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad \textit{Equation 5.2}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad \textit{Equation 5.3}$$

$$F1 - score\ = \frac{2 * (Precision * Recall)}{Precision + Recall} \quad \textit{Equation 5.4}$$

Table 5.1 and *Figure 5.1* contains the accuracy, precision, recall, and F1-score obtained for all original datasets using all ML and DL techniques. As we can see, the highest-performing algorithms are *Deep Neural Network*, *Decision Tree*, and *Random Forest* classifiers. They obtained 99% to 98% accuracies for all three datasets. Also, the precision, recall, and F-1 scores

are 97-98%. *Multi-Layer Perceptron (MLP)* showed promise with 98% accuracy for all datasets and 96-97% precision and recall. *Support Vector Machine* with *'rbf' kernel* worked well with all the datasets with 99-98% precision, recall, and accuracy. However, *SVM* with a *linear kernel* worked better with the smallest dataset (97-98%) but showed deteriorated performance for the bigger datasets (94-96%). *Logistic Regression* and *Naïve Bias* followed a similar trend, with good results for the small dataset (95-98%) and relatively lower results for bigger datasets (92-96%). Primarily, *Naïve Bias* provided accuracy of 80% for the largest dataset and precision, recall, and F1 scores in the 72-84% range. *Quadratic Discriminant Analysis (QDA)* displayed high accuracies for all datasets (94-99%) but relatively lower precision, recall, and F1 scores, especially for the larger datasets (in the range of 90-95%). On the other hand, *Gradient Boosting* performed well for larger datasets with accuracy, precision, recall, and F1 scores in the range of 96-98% but showed lower results for the smallest dataset (84-95%). Contrary to all these, the *AdaBoost* algorithm does not seem to work well for any of the datasets seemingly predicting randomly. However, the performance is a little better for larger datasets.

Table 5. 1: Experimental results- Accuracy, Precision and Recall on Original dataset

Algorithms		Small dataset – 100 nodes	Medium dataset – 200 nodes	Large dataset – 500 nodes
Logistic Regression	Accuracy	0.98	0.96	0.94
	Precision	0.97	0.95	0.93
	Recall	0.97	0.94	0.92
	F1 score	0.97	0.94	0.93
Naïve bias	Accuracy	0.96	0.92	0.8
	Precision	0.96	0.93	0.72
	Recall	0.95	0.92	0.84
	F1 score	0.96	0.92	0.74
Support Vector Machine-Linear	Accuracy	0.98	0.96	0.96
	Precision	0.98	0.96	0.95
	Recall	0.97	0.94	0.94
	F1 score	0.98	0.95	0.94
Support Vector Machine-Polynomial	Accuracy	0.98	0.98	0.98
	Precision	0.98	0.98	0.97
	Recall	0.97	0.96	0.97
	F1 score	0.98	0.97	0.97

Decision Tree	Accuracy	0.99	0.98	0.98
	Precision	0.98	0.97	0.97
	Recall	0.98	0.97	0.97
	F1 score	0.98	0.97	0.97
Random Forest	Accuracy	0.99	0.98	0.98
	Precision	0.99	0.98	0.97
	Recall	0.97	0.97	0.97
	F1 score	0.98	0.97	0.97
AdaBoost	Accuracy	0.27	0.32	0.55
	Precision	0.14	0.19	0.43
	Recall	0.13	0.25	0.39
	F1 score	0.07	0.18	0.39
Gradient Boosting	Accuracy	0.95	0.98	0.98
	Precision	0.84	0.98	0.97
	Recall	0.85	0.97	0.96
	F1 score	0.84	0.98	0.96
Multi-Layer Perceptron (MLP)	Accuracy	0.98	0.98	0.98
	Precision	0.98	0.97	0.96
	Recall	0.97	0.97	0.96
	F1 score	0.98	0.97	0.96
Quadratic discriminant analysis	Accuracy	0.99	0.94	0.95
	Precision	0.94	0.9	0.93
	Recall	0.95	0.95	0.95
	F1 score	0.95	0.92	0.94
Deep Neural Network	Accuracy	0.99	0.98	0.98
	Precision	0.98	0.98	0.97
	Recall	0.96	0.97	0.97
	F1 score	0.97	0.97	0.97

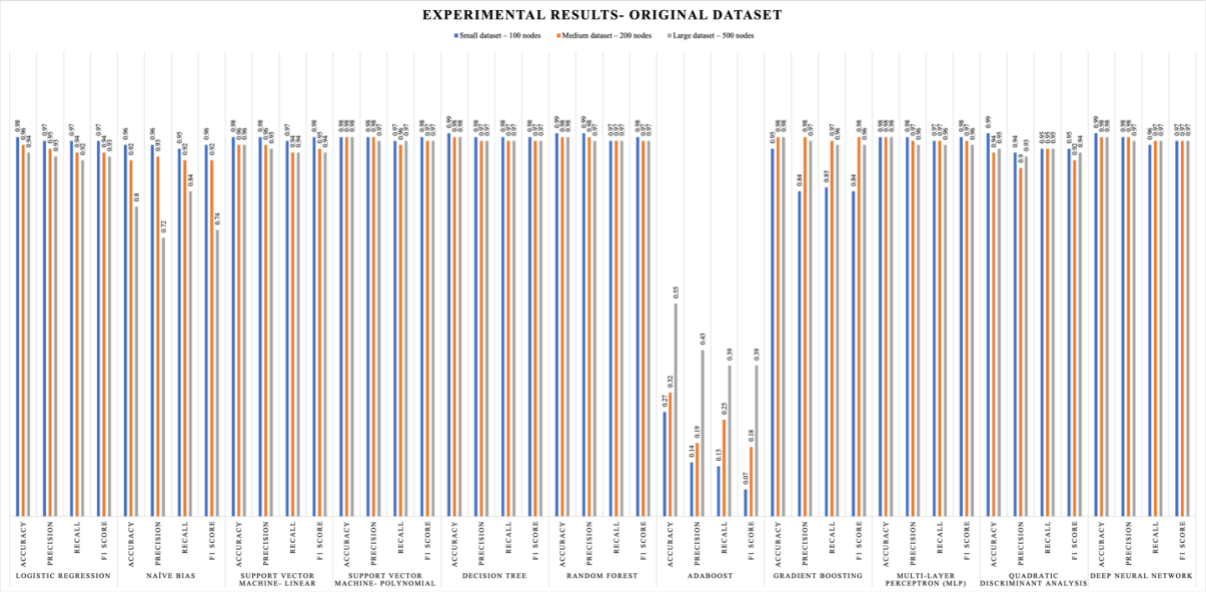

Figure 5. 1: Experimental results- Accuracy, Precision and Recall on Original dataset

Figures 5.2, 5.4, and 5.6 show the confusion matrix for all the datasets with the DNN model. As we can observe, the card failure category is often misclassified with some other categories. This could be because card failure happens due to contributing factors from other failure categories. *Figures 5.3, 5.5, and 5.7* show consecutively the learning and accuracy curves for all the datasets. As we can see, there is a steady growth in accuracy and a decreasing loss. The validation curve is also going smoothly with the training curve.

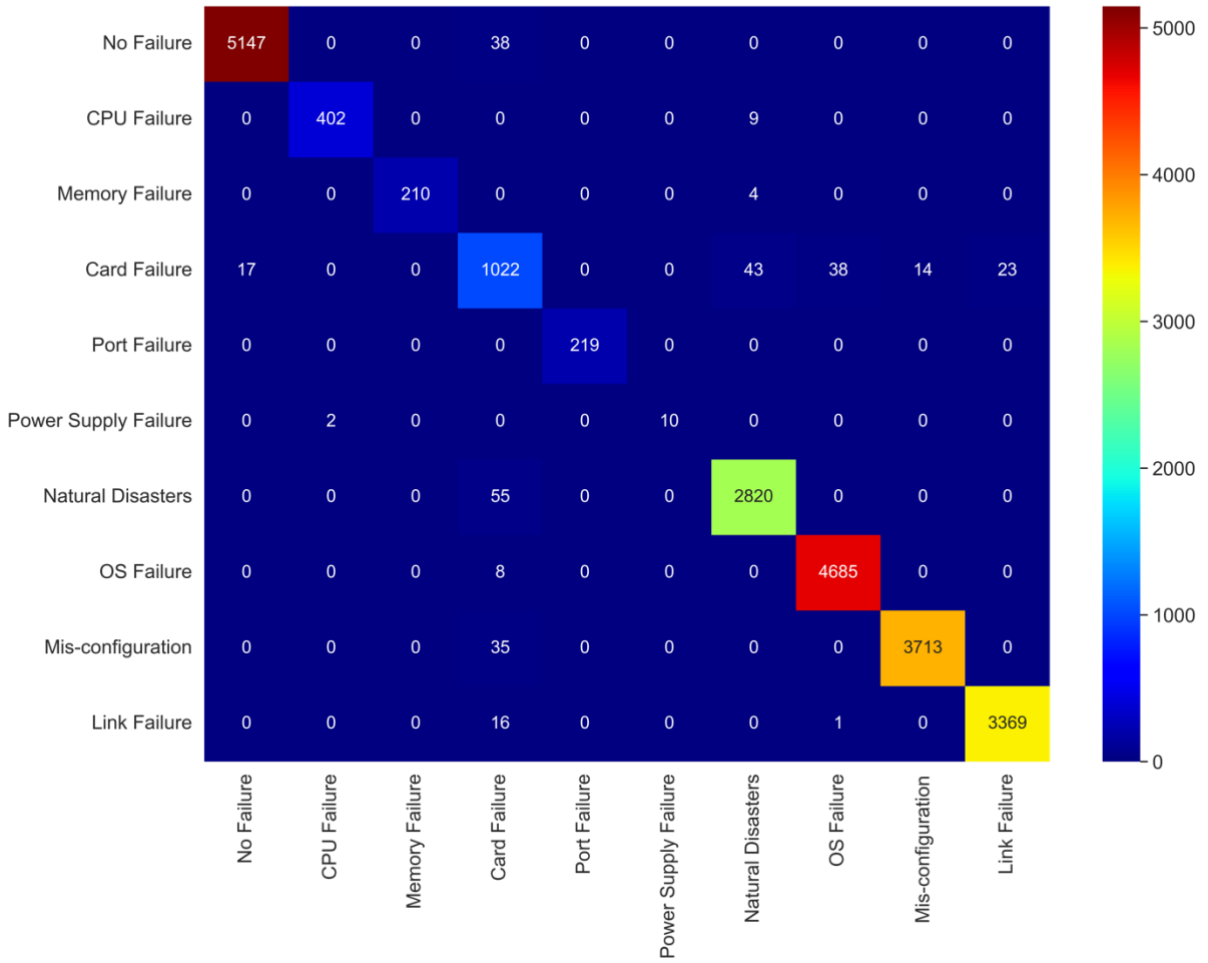

Figure 5. 2: Confusion Matrix for DNN (100 nodes)

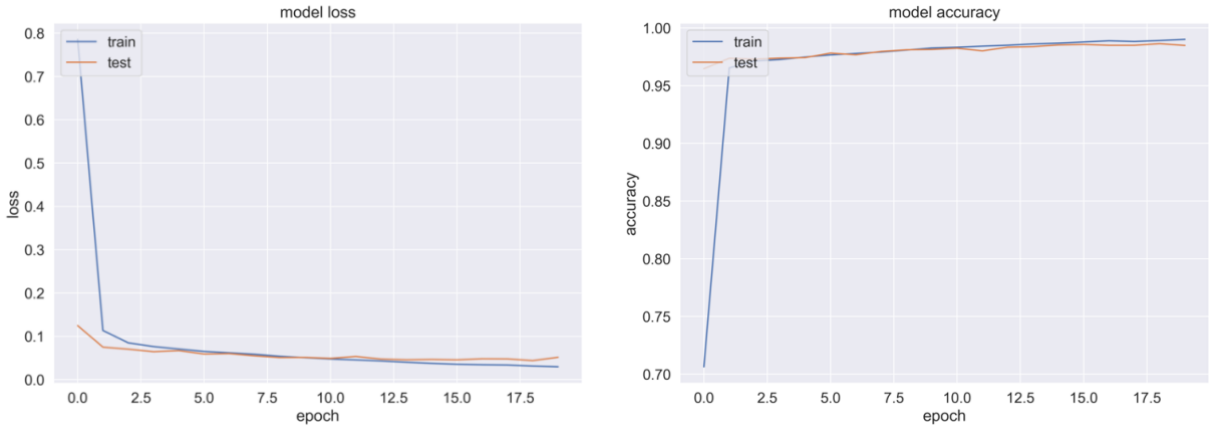

Figure 5. 3: Accuracy curve and Learning curve (100 nodes)

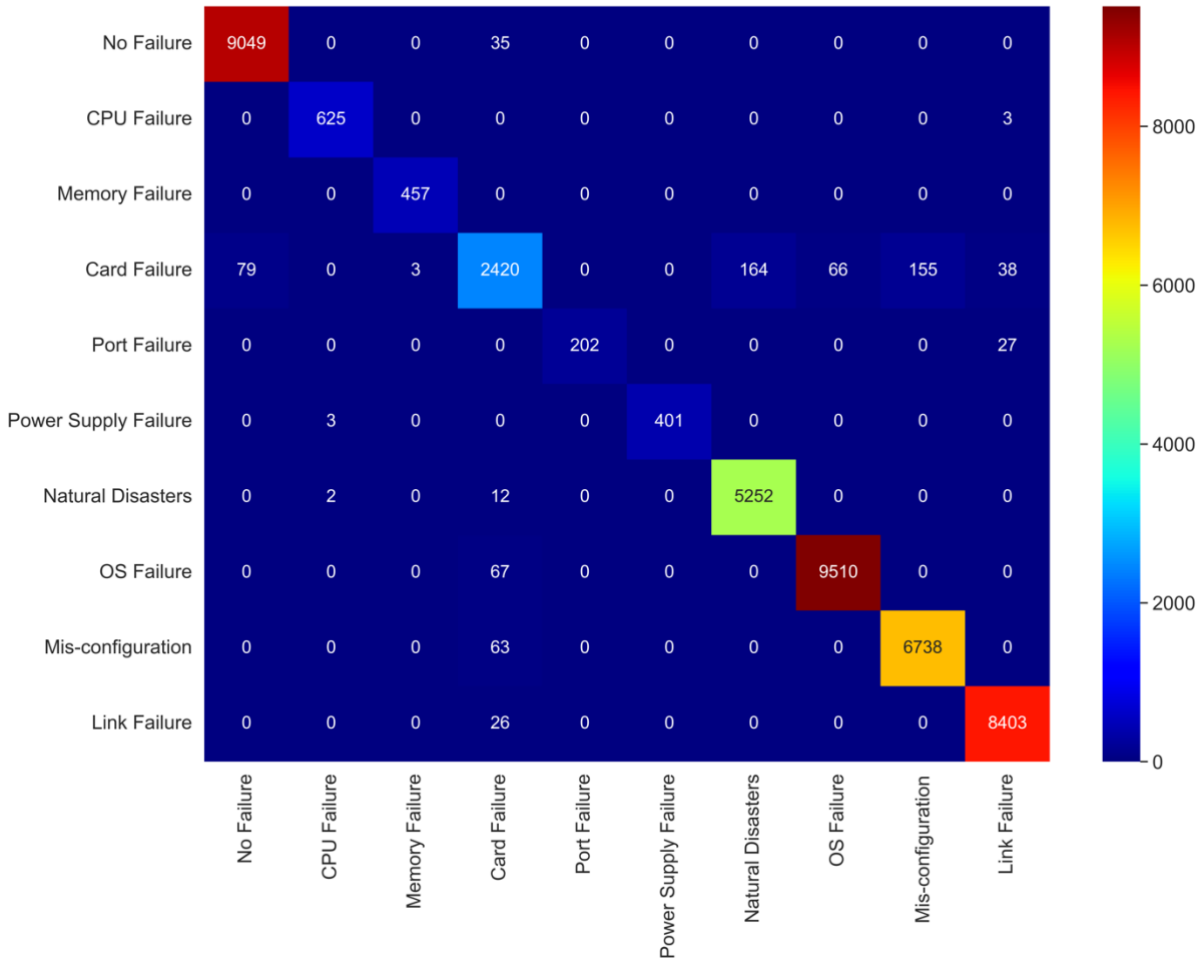

Figure 5. 4: Confusion matrix for DNN - 200 nodes

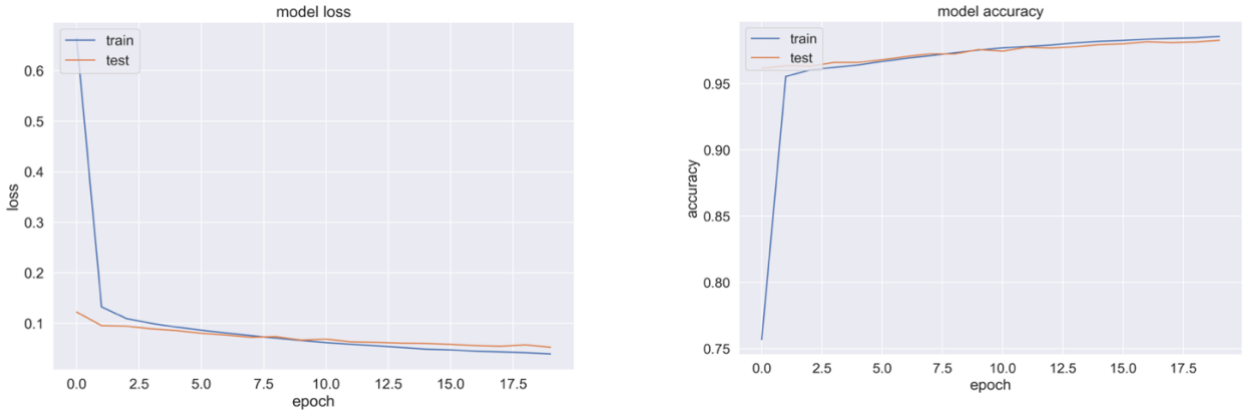

Figure 5. 5: Accuracy and loss curve - 200 nodes

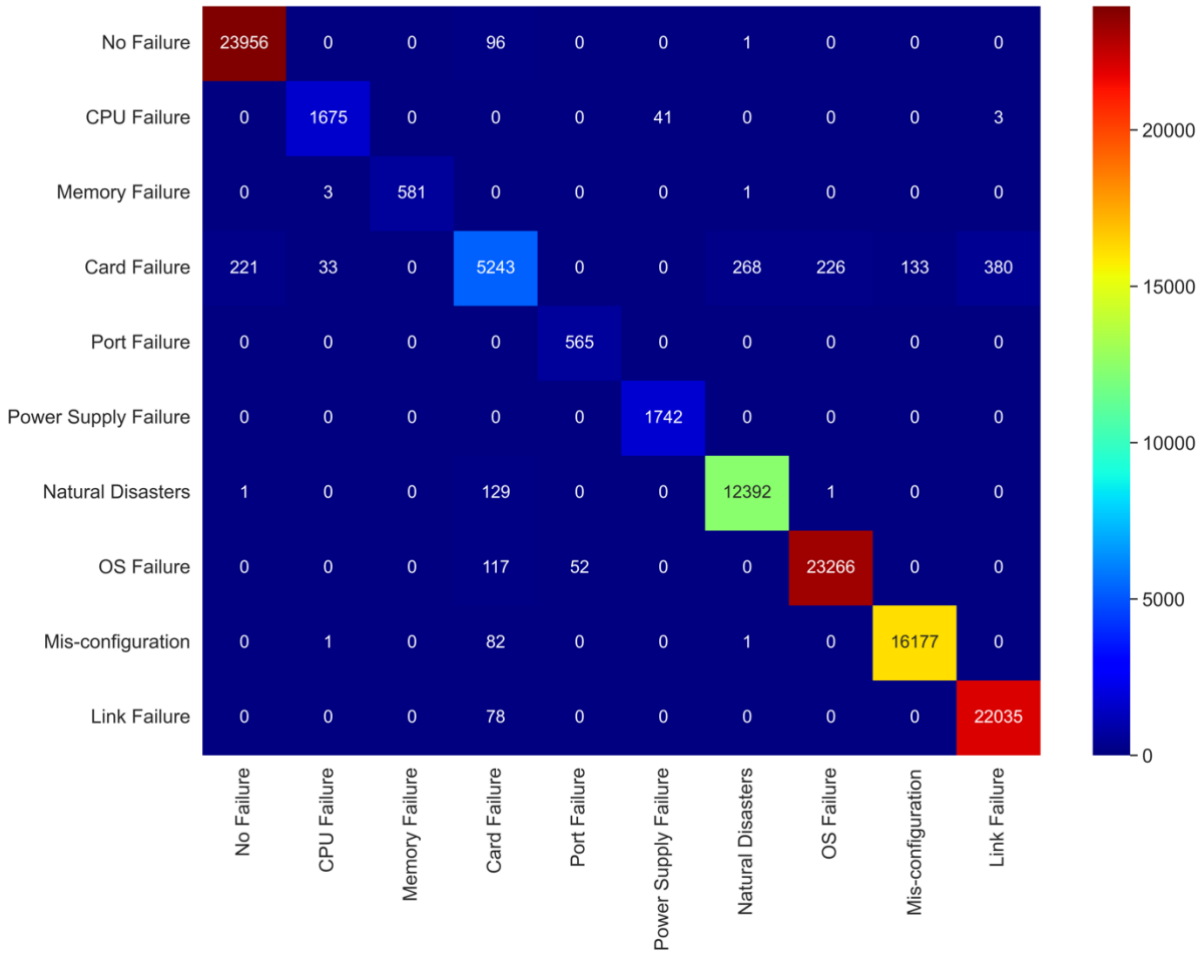

Figure 5. 6: Confusion matrix- 500 nodes

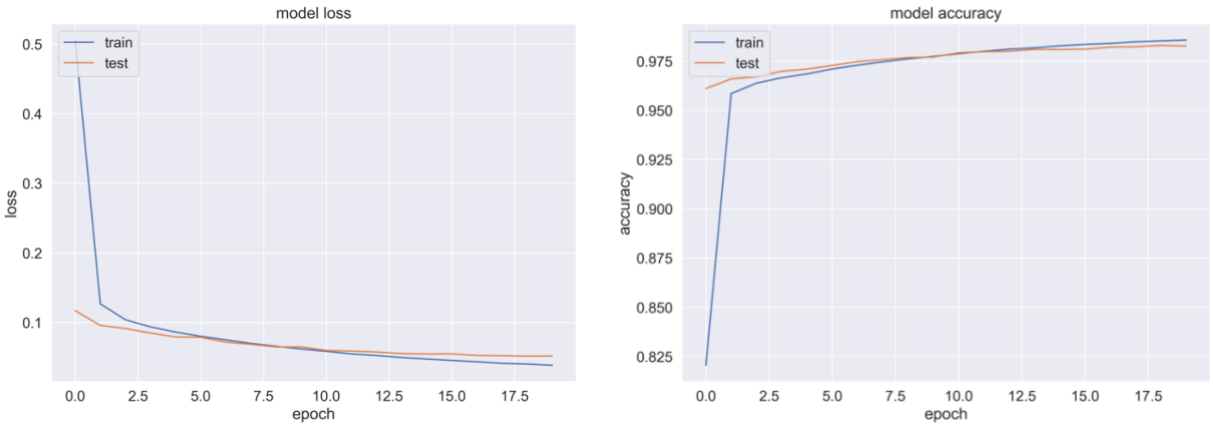

Figure 5. 7: Accuracy and learning curves - 500 nodes.

Table 5. 2: Time and Memory consumption for original dataset

Algorithms		Small dataset – 100 nodes	Medium dataset – 200 nodes	Large dataset – 500 nodes
Logistic Regression	Time (s)	37.21	71.98	194.34
	Memory (Mib)	3604.33	3861.02	6225.62
Naïve bias	Time (s)	0.26	0.4	1.41
	Memory (Mib)	2072.91	2272.61	4283.36
SVM- Linear	Time (s)	66.34	1107.35	59791.3
	Memory (Mib)	2870.33	3625.36	7977.5
SVM- Polynomial	Time (s)	85.87	2600.78	5125.92
	Memory (Mib)	2801.86	2962.95	5074.8
Decision Tree	Time (s)	3.52	7.82	28.21
	Memory (Mib)	2616.81	3211.25	3770.39
Random Forest	Time (s)	15.21	34.48	112.68
	Memory (Mib)	2618.25	2321.34	4707.55
AdaBoost	Time (s)	13.13	25.34	78.33
	Memory (Mib)	2153.52	2796.48	5500.77
Gradient Boosting	Time (s)	669.36	1324.91	3601.29
	Memory (Mib)	1742.11	2448.33	3968.92
MLP	Time (s)	80.76	164.55	684.48
	Memory (Mib)	2565.19	3845.83	4009.84
QDA	Time (s)	0.8	1.48	3.55
	Memory (Mib)	2224.31	3336.91	4894.22
Deep Neural Network	Time (s)	843.81	1640.51	4734.67
	Memory (Mib)	1268.19	1441.85	2174.32

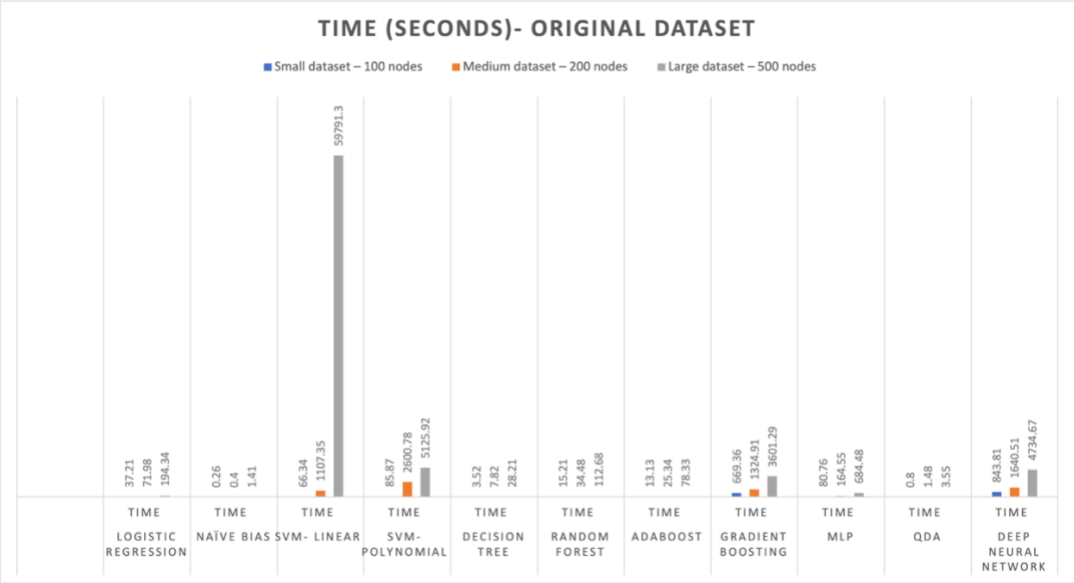

Figure 5. 8: Time consumption for original dataset

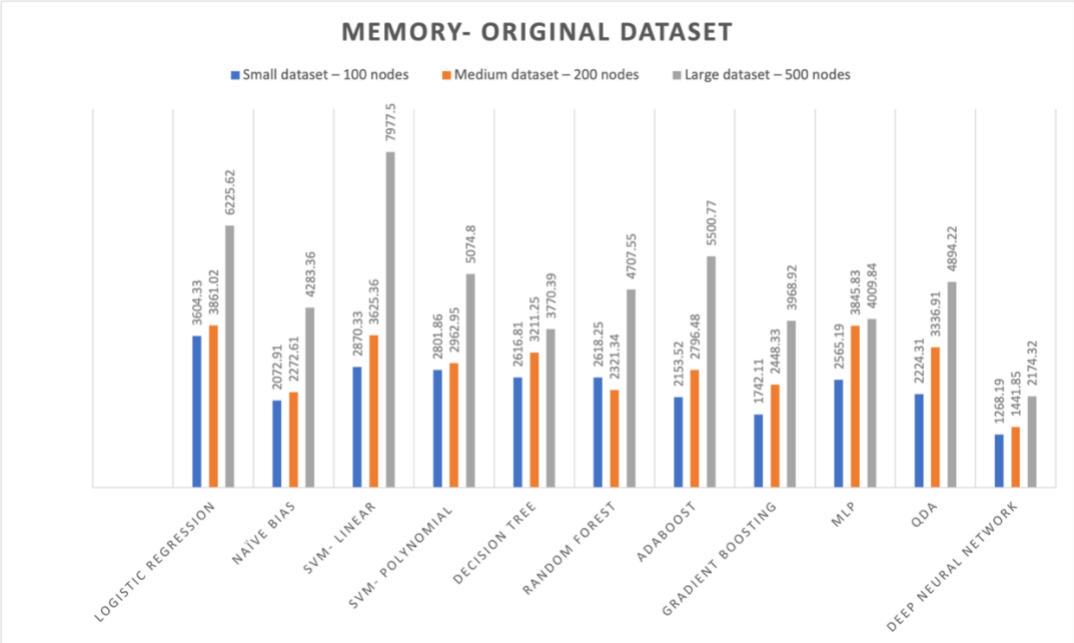

Figure 5. 9: Memory consumption for original dataset

As we can see in *Table 5.2, Figure 5.8, and 5.9*, for the smallest dataset, *Naive Bias, QDA, DT, AdaBoost*, and *RF* took the least time to train the model. *Logistic regression, SVM*, and *MLP* took a moderate amount of time. *Gradient boosting* and *DNN* took the longest to train. Additionally, the time to train the model for larger datasets demonstrates a linear increase except for SVM. Especially with a linear kernel, SVM took longer to train the two large datasets, surpassing gradient boosting and DNN. Now, if we look at memory consumption, all techniques use approximately the same kind of memory without any significant variation. The memory usage for the large datasets follows a linear trend for all models.

5.2 Experimental Results, Time-Space: SMOTE

Table 5.3 and Figure 5.10 shows all datasets accuracy, precision, and recall after applying SMOTE on the original training data for all the ML and DL techniques. Like the original dataset, *Deep Neural Network*, *Decision Tree*, and *Random Forest* classifiers are the highest-performing algorithms. However, they obtained slightly better results for the two large datasets. *MLP* obtained almost the same results as the original. There is a slight increase in *SVM* with the *'rbf' kernel*, while *SVM* with *linear kernel* showed similar results. *Logistic Regression* performed better for the medium dataset. *Naive Bias, AdaBoost*, and *QDA* showed similar performances. The *Gradient Boosting* algorithm performed much better with the SMOTE applied to the small dataset with 99% accuracy, precision, and recall.

Table 5. 3: Experimental results- Precision, Recall, Accuracy with SMOTE

Algorithms		**Small dataset – 100 nodes**	**Medium dataset – 200 nodes**	**Large dataset – 500 nodes**
Logistic Regression	Accuracy	0.97	0.95	0.92
	Precision	0.95	0.91	0.91
	Recall	0.97	0.95	0.93
	F1 score	0.96	0.93	0.92
Naïve bias	Accuracy	0.96	0.92	0.77
	Precision	0.96	0.92	0.72
	Recall	0.95	0.92	0.83
	F1 score	0.96	0.92	0.74
Support Vector	Accuracy	0.97	0.95	0.94
	Precision	0.96	0.92	0.93

Machine-Linear	Recall	0.97	0.95	0.95
	F1 score	0.96	0.93	0.94
Support Vector Machine-Polynomial	Accuracy	0.99	0.98	0.98
	Precision	0.99	0.95	0.97
	Recall	0.98	0.97	0.98
	F1 score	0.98	0.96	0.97
Decision Tree	Accuracy	0.99	0.98	0.98
	Precision	0.97	0.97	0.97
	Recall	0.98	0.97	0.97
	F1 score	0.98	0.97	0.97
Random Forest	Accuracy	0.99	0.98	0.98
	Precision	0.98	0.98	0.97
	Recall	0.98	0.97	0.97
	F1 score	0.98	0.97	0.97
AdaBoost	Accuracy	0.03	0.02	0.36
	Precision	0.06	0.01	0.42
	Recall	0.26	0.23	0.34
	F1 score	0.07	0.02	0.27
Gradient Boosting	Accuracy	0.99	0.98	0.98
	Precision	0.99	0.97	0.96
	Recall	0.99	0.97	0.97
	F1 score	0.99	0.97	0.96
Multi-Layer Perceptron (MLP)	Accuracy	0.98	0.98	0.98
	Precision	0.98	0.96	0.96
	Recall	0.98	0.96	0.96
	F1 score	0.98	0.96	0.96
Quadratic discriminant analysis	Accuracy	0.96	0.95	0.95
	Precision	0.85	0.91	0.93
	Recall	0.84	0.95	0.95
	F1 score	0.84	0.93	0.94
Deep Neural Network	Accuracy	0.99	0.98	0.98
	Precision	0.98	0.96	0.97
	Recall	0.98	0.97	0.98
	F1 score	0.98	0.96	0.97

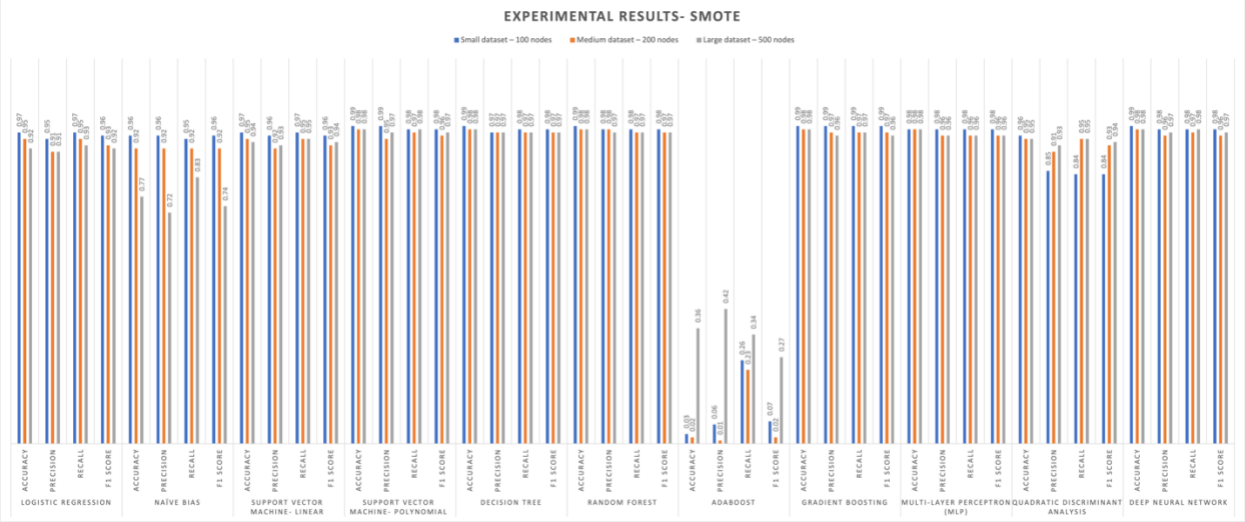

Figure 5. 10: Experimental results- Precision, Recall, Accuracy with SMOTE

Figures 5.11, 5.13, and 5.15 are the confusion matrices for all the SMOTE datasets with the DNN model. If we compare with the original dataset results, we can observe a visible improvement over the card failure category, especially for bigger datasets. The correctly recognized card failure numbers were: 1022, 2420, and 5243, respectively, with the original dataset. With SMOTE, these numbers increased to 1039, 2623, and 5769. *Figures 5.12, 5.14, and 5.16* contain the learning and accuracy curves, which showed a similar trend to the original dataset.

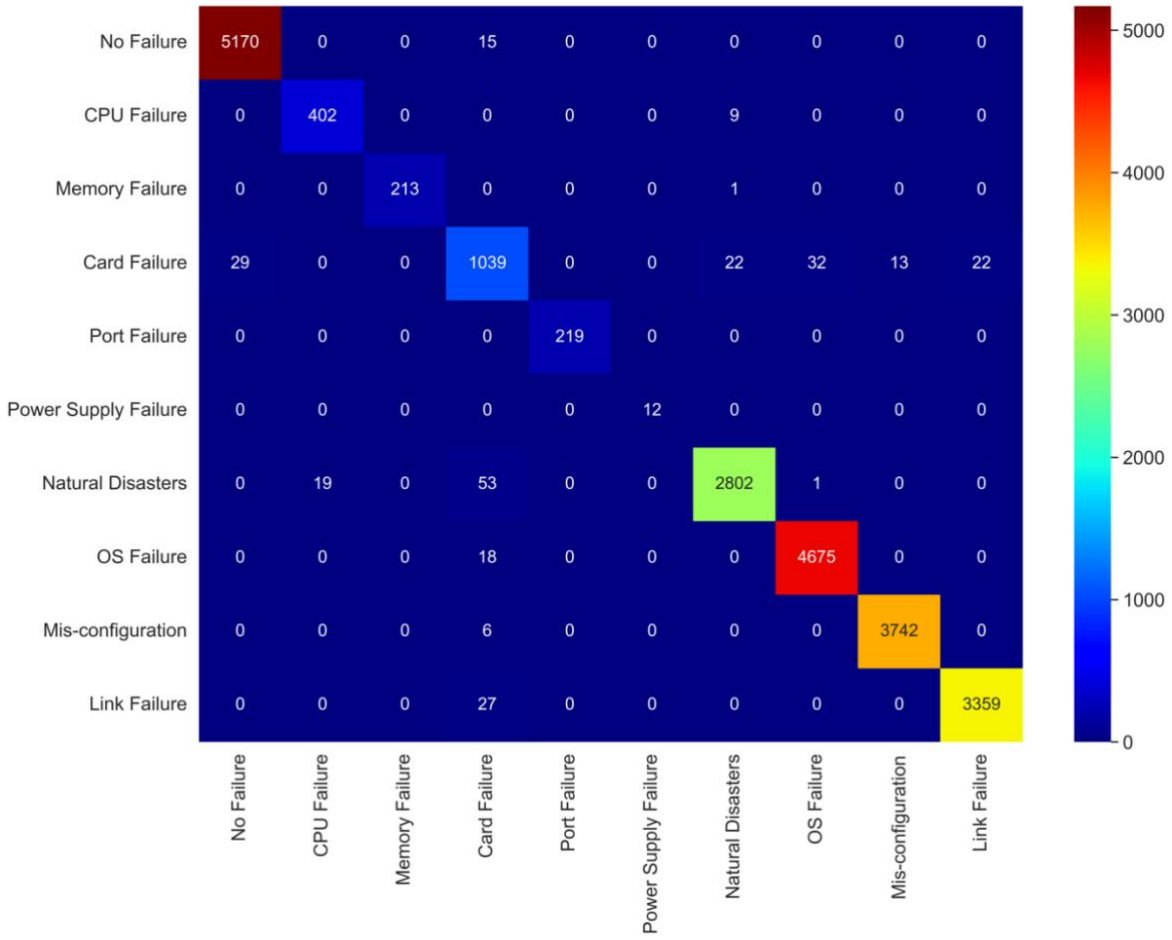

Figure 5. 11: Confusion matrix (SMOTE)- 100 nodes.

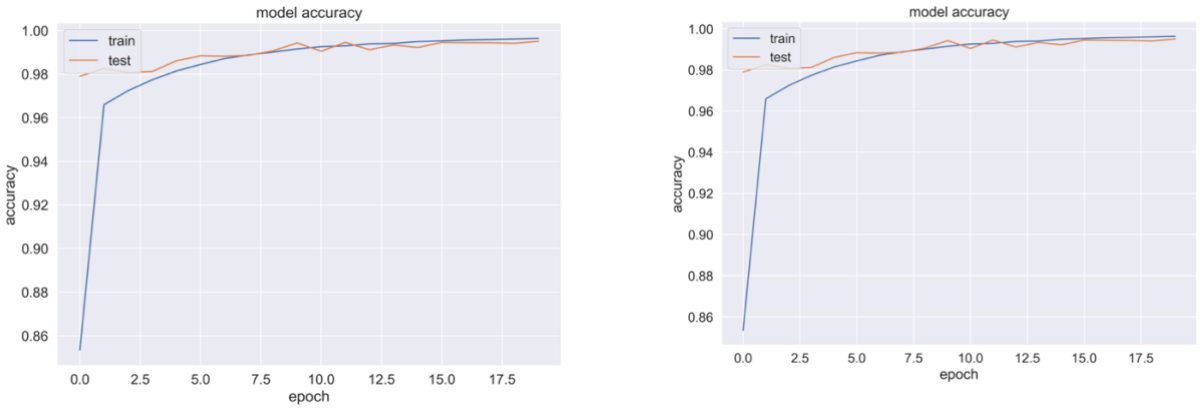

Figure 5. 12: Accuracy and learning curve (SMOTE) - 100 nodes.

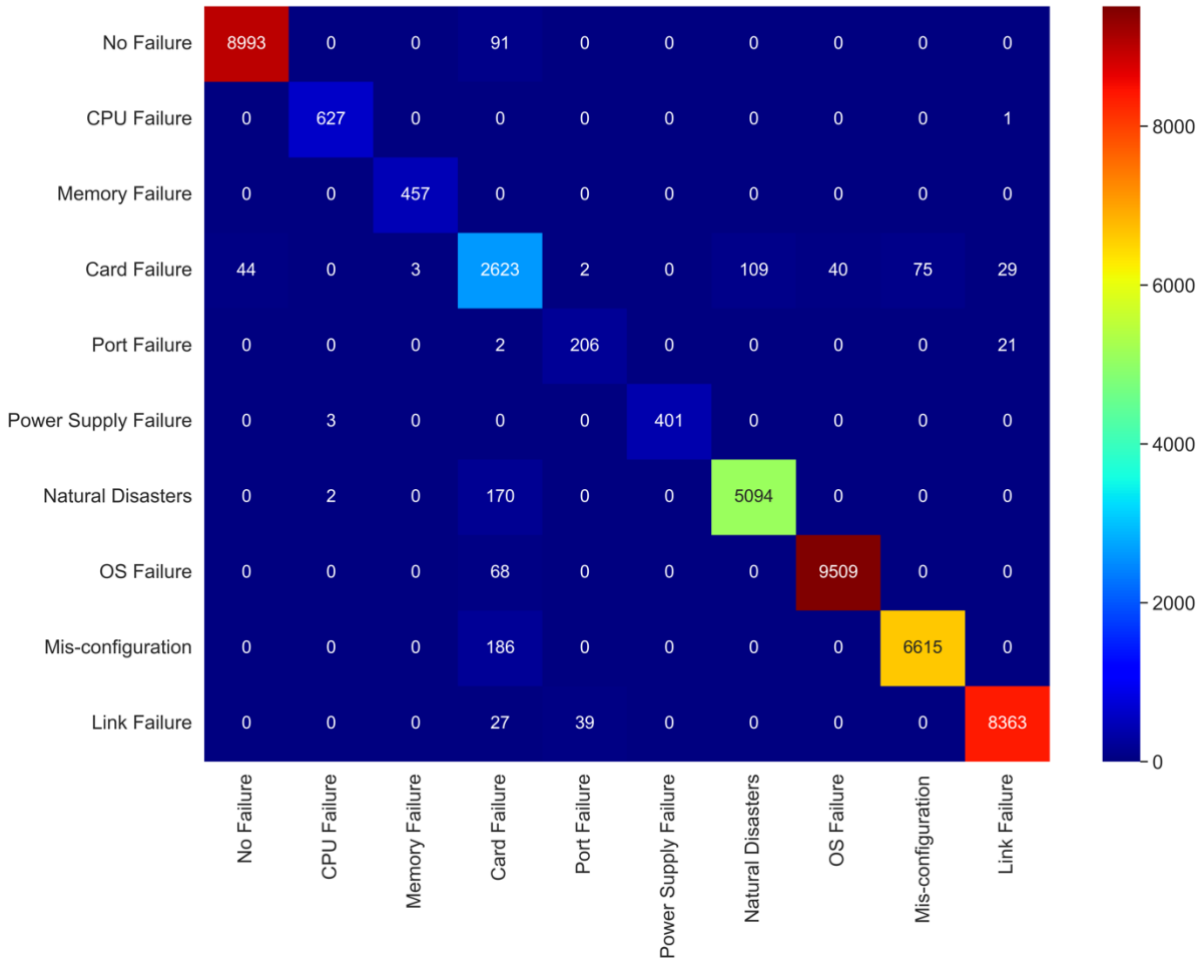

Figure 5. 13: Confusion matrix (SMOTE)- 200 nodes

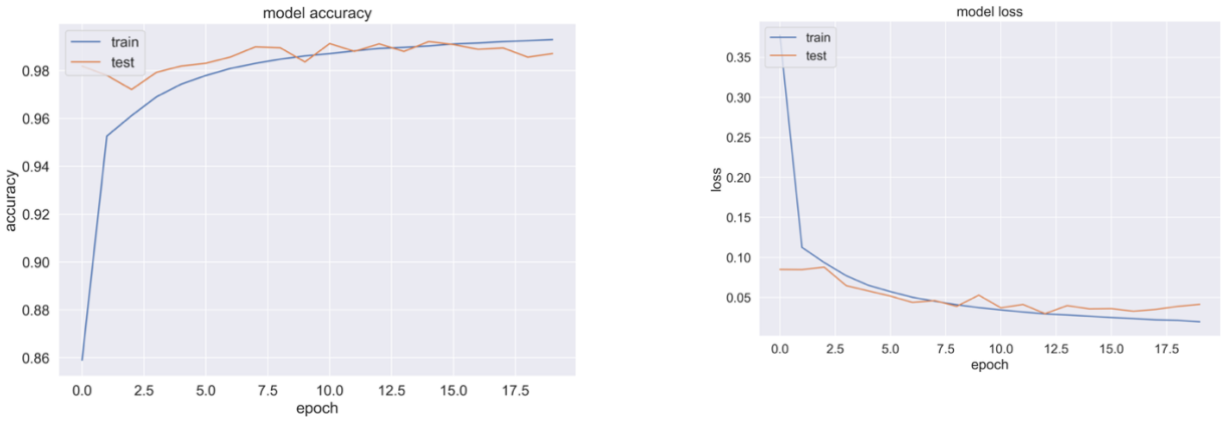

Figure 5. 14: Accuracy and learning curve- 200 nodes.

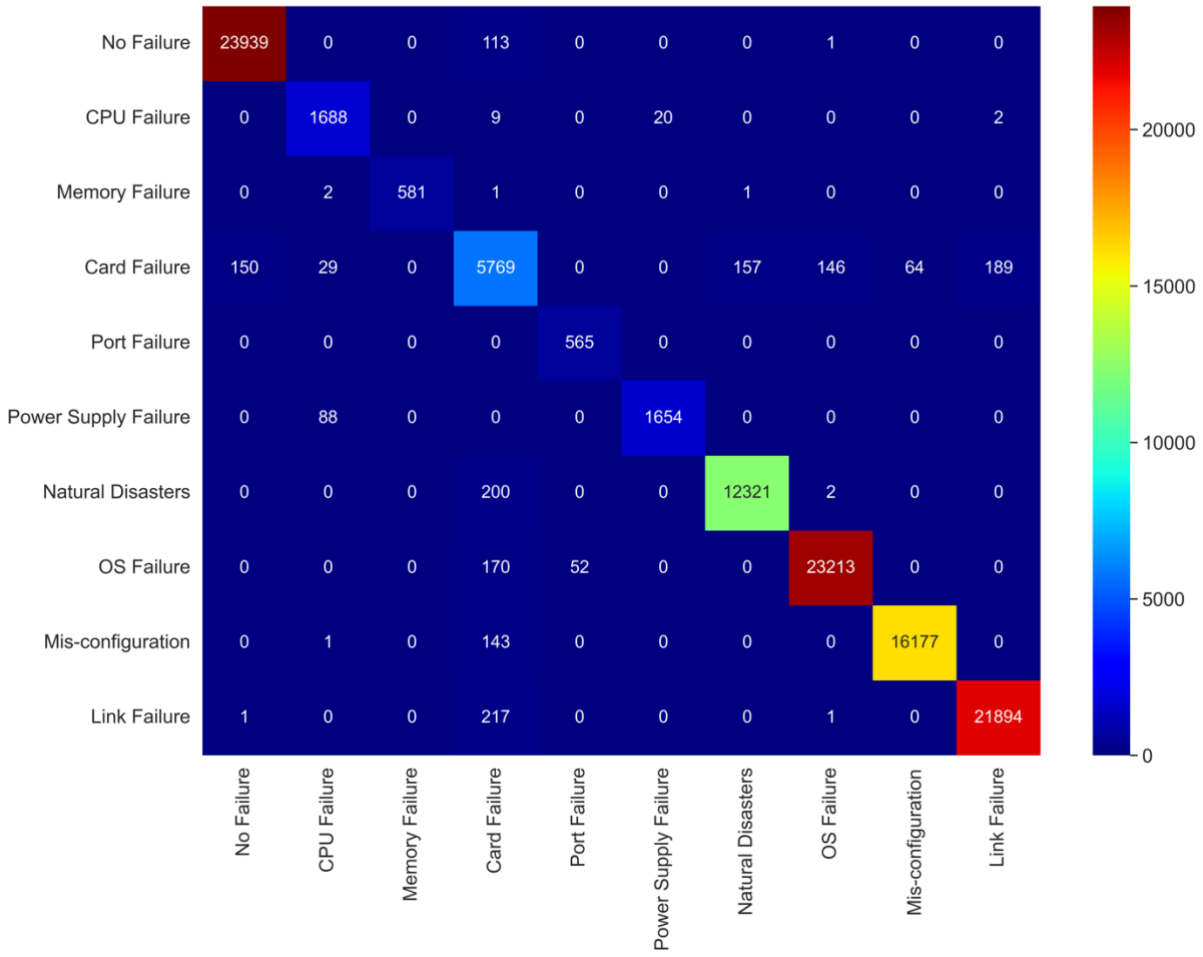

Figure 5. 15: Confusion matrix (SMOTE) - 500 nodes

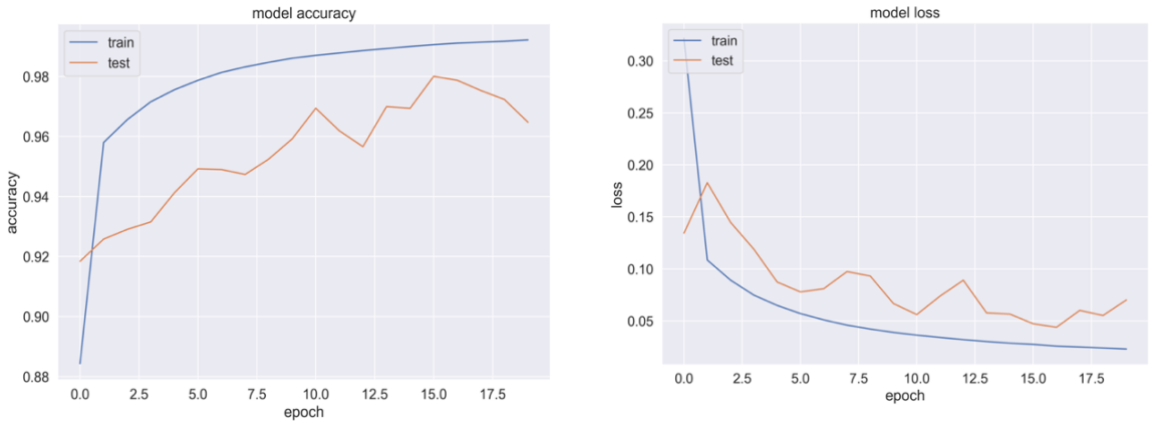

Figure 5. 16: Accuracy and learning curves - 500 nodes.

Table 5. 4: Time and memory consumption with SMOTE

Algorithms		Small dataset – 100 nodes	Medium dataset – 200 nodes	Large dataset – 500 nodes
Logistic Regression	Time (s)	87.09	162.1	476.42
	Memory (Mib)	4661.02	7030.39	8325.73
Naïve bias	Time (s)	0.43	1.11	4.88
	Memory (Mib)	2372.97	4161.11	5704.41
SVM- Linear	Time (s)	233.51	3320.69	120896.97
	Memory (Mib)	3739.44	5436.53	8667.52
SVM- Polynomial	Time (s)	138.59	1130.1	8142.42
	Memory (Mib)	3915.64	5186.91	7389.3
Decision Tree	Time (s)	10.8	28.11	101.87
	Memory (Mib)	2672.88	4384.69	8272.14
Random Forest	Time (s)	66.4	150.14	568.71
	Memory (Mib)	2950.72	4620.59	8084.34
AdaBoost	Time (s)	77.81	152.28	491.48
	Memory (Mib)	2861.41	4601.19	8083.3
Gradient Boosting	Time (s)	3791.42	7040.16	21079.75
	Memory (Mib)	2705.91	5252.14	7870.41
MLP	Time (s)	79.59	261.6	1211.37
	Memory (Mib)	3547.33	5120.5	6566.3
QDA	Time (s)	1.51	2.52	6.76
	Memory (Mib)	3292.33	4654.77	7957.67
Deep Neural Network	Time (s)	1725.22	4656.34	8133.9
	Memory (Mib)	1760.8	2374.35	3931.74

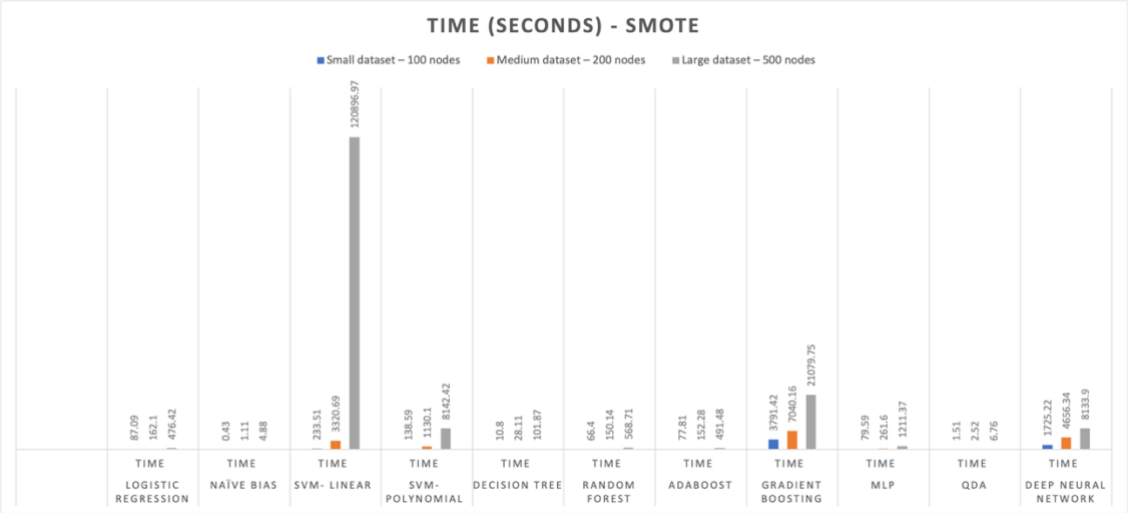

Figure 5. 17: Time consumption with SMOTE

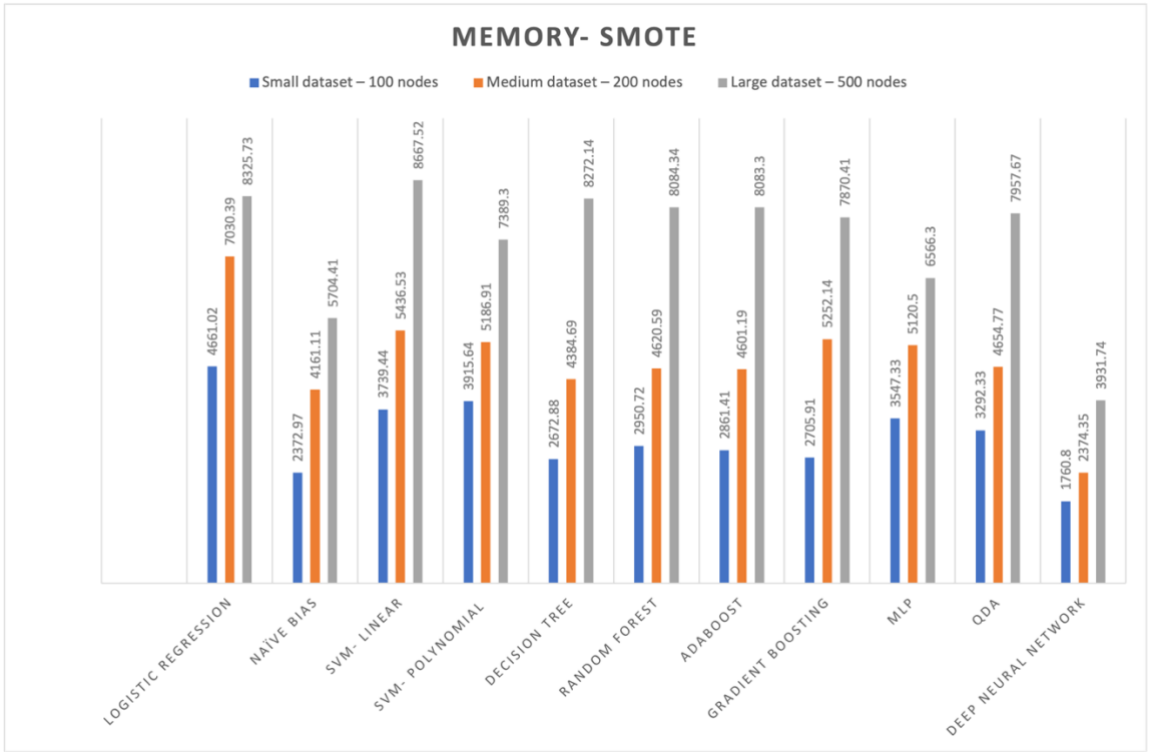

Figure 5. 18: Memory consumption with SMOTE

Table 5.4, Figure 5.17, Figure 5.18 shows the time and memory consumption with the datasets after applying SMOTE. They demonstrate a similar trend as before. After applying SMOTE, overall data points for training sets increased, and the time and memory requirements increased proportionally.

5.3 Experimental Results for Failure Prediction

Each node is considered separately for each dataset, and the proposed model is trained separately for each node. For the failure prediction, a separate model for each node of one dataset is trained consecutively, and later the average prediction error is calculated for all the nodes. Therefore, there are 100 models for the small dataset, 200 for the medium dataset, and 500 for the large dataset. The average results of these models are later calculated and reported. Each model is trained for 10 epochs with a learning rate 0.0001 and Adam optimizer. The Root Mean Square Error (RMSE) is considered the evaluation matrix. To calculate the RMSE, the square root distance of the predicted and actual sequence is taken (*Equation 5.5*).

$$RMSE = \sqrt{\frac{(Predicted - Actual)^2}{Number\ of\ Samples}} \qquad \textit{Equation 6}$$

Table 5. 5: Average RMSE for all datasets

DataSet	RMSE
Small dataset – 100 nodes	1.0381156792830557
Medium dataset – 200 nodes	0.9278651685828186
Large dataset – 300 nodes	0.9751401700710098

The RMSE average of all dataset nodes is presented in *Table 5.5*. As can be seen, the average error rate is relatively low. With more training data, the error ratings for the larger datasets are lower.

Chapter 6

6 Conclusion and Future work

Evidentially, our daily activities are heavily impacted by network failures owing to external elements like software, hardware, broken links, misconfiguration, adverse weather, and other concerns. Network failure during medical and personal security emergencies can have catastrophic consequences. In addition, time-sensitive general or industrial operations suffer from network disruptions. This research seeks to develop a predictive model to correctly identify failure categories and aid ISPs in avoiding misusing redundant resources.

A comprehensive investigation of network failure case studies, available datasets, and existing tools and techniques revealed a need for public datasets on network failure that can be utilized for various purposes. The vast majority of current data sets are log-based or domain-specific. In addition, most recent research focuses on fiber cut location prediction and alarm log-based domain-specific prediction. This study proposed a simulation-based approach with three prototype network architectures of varying sizes. These networks capture network failure data following Cisco guidelines [2]. Even though the generated dataset is examined to ensure its veracity, it is challenging to follow real-world data's exact distribution due to the numerous interconnected components.

Subsequently, these datasets are trained and evaluated using advanced Machine Learning and Deep Learning techniques. The optimal outcome demonstrated 99% accuracy, 98% precision, 96% recall, and 97% F1 score. As the failure data is inherently skewed, SMOTE generates synthetic samples of minority classes. Using SMOTE significantly improved the recognition accuracy of certain minority classes. In addition, each network node's availability data is used to train an LSTM model for failure sequence prediction. The low average root mean square error rates in sequence prediction are encouraging. Still, there is a need for a more comprehensive analysis of the efficacy of the sequence model on each node of the network separately.

This study is a proof of concept of a network failure prediction system. This method can serve as a baseline for further research with actual failure data. To deploy the proposed identification and prediction model for an ISP, we need to get a better understanding of their network architecture, components, and monitoring systems. Based on that information, the

proposed framework can be integrated with the central monitoring system of the ISP. This approach will provide a centralized system for real-time failure identification and prediction based on the incoming network data. Alternatively, we can deploy the model on distributed edge nodes for quicker response time as the processing will be done locally in each node. This approach provides flexibility in the case of a very complex network. After the model is deployed it will then use an online learning algorithm to update the model in real time with the inclusion of new data. This will particularly be advantageous for dynamic and rapidly changing network conditions. Some modifications and adjustments will be necessary to the proposed model parameters for Software Defined Network (SDN) which can be addressed in future works. However, applying this model in real-world scenarios can increase internet availability to ensure the uninterrupted delivery of time-critical operations. In addition, ISPs can reduce the money lost due to network failure.

References

[1] Wang, S. S., & Franke, U. (2020). Enterprise IT service downtime cost and risk transfer in a supply chain. *Operations Management Research*, *13*(1-2), 94-108.

[2] Hussain, I. (2005). Understanding High Availability of IP and MPLS Networks. *white paper, Ciscopress*.

[3] Alriksson, F., Boström, L., Sachs, J., Wang, Y. P. E., & Zaidi, A. (2020). Critical IoT connectivity Ideal for Time-Critical Communications. *Ericsson technology review*, *2020*(6), 2-13.

[4] SecurityScorecard. (2023, February 2). *What is the CIA Triad? Definition, Importance, & Examples*. https://securityscorecard.com/blog/what-is-the-cia-triad/

[5] Wikipedia contributors. (2023, January 14). *2022 Rogers Communications outage*. Wikipedia. https://en.wikipedia.org/wiki/2022_Rogers_Communications_outage

[6] Moozakis, C. (2021, June 28). *Five-nines availability: What it really means*. Networking. https://www.techtarget.com/searchnetworking/feature/The-Holy-Grail-of-five-nines-reliability

[7] Stallings, W. (2019). Pearson etext cryptography and network security: Principles and practice -- access card (8th ed.). Pearson.

[8] IBM Documentation. (n.d.). https://www.ibm.com/docs/es/spss-modeler/18.0.0?topic=networks-nodal-degree

[9] Canadian Press. (2015, July 25). https://saultonline.com/2015/07/rogers-outage-disrupts-service-across-canada/

[10] Hardy, I. (2015, July 26). Rogers experiencing network outage and mainly impacting several provinces [Update]. MobileSyrup. https://mobilesyrup.com/2015/07/25/rogers-experiencing-network-outage-and-mainly-impacting-those-ontario-british-columbia-and-alberta/

[11] Bianchini, E. (2019, July 10). Rogers outage: Can you be compensated for wireless service disruptions? Yahoo Finance. https://ca.finance.yahoo.com/news/rogers-freedom-mobile-fido-chatr-outage-200842933.html

[12] O'Neil, L. (2019, July 8). *Massive Rogers wireless outage causes total chaos in Toronto*. blogTO. https://www.blogto.com/tech/2019/07/massive-rogers-wireless-outage/

[13] Munarriz, R. (2016, February 16). *Comcast Outage Comes at a Terrible Time*. The Motley Fool. https://www.fool.com/investing/general/2016/02/16/comcast-outage-comes-at-a-terrible-time.aspx

[14] King, H. (2016, February 17). *Comcast offers credit for Monday's service outage*. CNNMoney. https://money.cnn.com/2016/02/17/technology/comcast-outage-credit/

[15] Seto, C., & Seto, C. (2020, August 7). *Bell, Telus still looking into cause of massive network outage*. therecord.com. https://www.therecord.com/news/waterloo-region/2020/08/07/bell-telus-still-looking-into-cause-of-massive-network-outage.html

[16] Lyons, K. (2020, August 30). *Cloudflare says its Sunday morning problems were due to CenturyLink outage*. The Verge. https://www.theverge.com/2020/8/30/21407429/cloudflare-down-websites-hulu-feedly-discord

[17] Prince, M. (2022, July 12). *August 30th 2020: Analysis of CenturyLink/Level(3) outage*. The Cloudflare Blog. https://blog.cloudflare.com/analysis-of-todays-centurylink-level-3-outage/

[18] Goldman, D. (2021, January 26). *East Coast hit with massive internet outage, disrupting remote work and Virtual School | CNN business*. CNN. Retrieved February 14, 2023, from https://www.cnn.com/2021/01/26/tech/verizon-fios-outage/index.html

[19] Peters, J. (2021, January 26). *Verizon says Fios internet should be returning to normal in the Northeast after disruptive outage*. The Verge. https://www.theverge.com/2021/1/26/22250571/verizon-fios-outage-problem-issue-east-coast-google-zoom

[20] *December 27, 2018 centurylink network outage report*. (n.d.). Retrieved February 15, 2023, from https://docs.fcc.gov/public/attachments/DOC-359134A1.pdf

[21] The Associated Press. (2018, December 28). *Nationwide internet outage affects CenturyLink customers*. CNBC. https://www.cnbc.com/2018/12/28/nationwide-internet-outage-affects-centurylink-customers.html

[22] Porter, J. (2018, December 7). *Millions of smartphones were taken offline by an expired certificate*. The Verge. https://www.theverge.com/2018/12/7/18130323/ericsson-software-certificate-o2-softbank-uk-japan-smartphone-4g-network-outage

[23] Graham-Cumming, J. (2022, August 4). *Details of the Cloudflare outage on July 2, 2019*. The Cloudflare Blog. https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/

[24] Jazeera, A. (2019, April 1). *Major US airlines hit by delays after technical glitch*. Aviation News | Al Jazeera. https://www.aljazeera.com/news/2019/4/1/major-us-airlines-hit-by-delays-after-technical-glitch

[25] *Summary of June 8 outage*. (2021, June 8). Fastly. https://www.fastly.com/blog/summary-of-june-8-outage

[26] Browne, R., & Shead, S. (2021, June 8). *What is Fastly and why did it just take a bunch of major websites offline?* CNBC. https://www.cnbc.com/2021/06/08/fastly-outage-internet-what-happened.html

[27] *Route leak causes global outage in level 3 network*. Digital Experience Monitoring. (n.d.). Retrieved February 14, 2023, from https://www.thousandeyes.com/blog/route-leak-causes-global-outage-level-3-network

[28] *Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/message/41926/

[29] *AWS Outage that Broke the Internet Caused by Mistyped Command*. (2017, March 2). Data Center Knowledge | News and Analysis for the Data Center Industry. https://www.datacenterknowledge.com/archives/2017/03/02/aws-outage-that-broke-the-internet-caused-by-mistyped-command

[30] Villas-Boas, A. (2019, July 9). *Major websites and services across the internet went down Tuesday because of a cloud network outage*. Business Insider. https://www.businessinsider.com/cloudflare-outage-causes-major-websites-across-internet-to-go-down-2019-7

[31] Moss, S. (2023, January 27). *Faulty ups at Telecity data center causes internet outages across the UK*. All Content RSS. Retrieved February 14, 2023, from https://www.datacenterdynamics.com/en/news/faulty-ups-at-telecity-data-center-causes-internet-outages-across-the-uk/

[32] *Bell services back online after "major" outage affecting phone, internet and TV service*. (2017, August 5). CBC. https://www.cbc.ca/news/canada/new-brunswick/bell-aliant-cellphone-landline-1.4235279

[33] McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. (1955). A proposal for the Dartmouth summer research project on artificial intelligence.

[34] Mitchell, T. (1997). Machine learning. McGraw Hill.

[35] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[36] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[37] *sklearn.linear_model.LogisticRegression*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[38] *1.9. Naive Bayes*. (n.d.). Scikit-learn. https://scikitlearn/stable/modules/naive_bayes.html

[39] *Support Vector Machines*. (n.d.). Scikit-learn. https://scikit learn/stable/modules/svm.html

[40] *sklearn.tree.DecisionTreeClassifier*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[41] *sklearn.ensemble.RandomForestClassifier*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[42] *sklearn.ensemble.AdaBoostClassifier*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[43] *sklearn.ensemble.GradientBoostingClassifier*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[44] *Neural network models (supervised)*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/neural_networks_supervised.html

[45] *Linear and Quadratic Discriminant Analysis*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/lda_qda.html

[46] Team, K. (n.d.). *Keras documentation: The Sequential class*. The Sequential Class. https://keras.io/api/models/sequential/

[47] Recurrent Neural Network (RNN) and LSTM. (2021b, November 22). Data Warehousing and Data Science. https://dwbi1.wordpress.com/2021/08/07/recurrent-neural-network- rnn-and-lstm/

[48] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[49] Labovitz, C., Ahuja, A., & Jahanian, F. (1999, June). Experimental study of internet stability and backbone failures. In *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 99CB36352)* (pp. 278-285).

[50] Hayford-Acquah, T., & Asante, B. (2017). Causes of fiber cut and the recommendation to solve the problem. *IOSR J. Electron. Commun. Eng*, *12*, 46-64.

[51] Nyarko-Boateng, O., Adekoya, A. F., & Weyori, B. A. (2021). Predicting the actual location of faults in underground optical networks using linear regression. *Engineering reports*, *3*(3), eng212304.

[52] Wang, Z., Zhang, M., Wang, D., Song, C., Liu, M., Li, J., ... & Liu, Z. (2017). Failure prediction using machine learning and time series in optical network. *Optics Express*, *25*(16), 18553-18565.

[53] Zhang, K., Xu, J., Min, M. R., Jiang, G., Pelechrinis, K., & Zhang, H. (2016, December). Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)* (pp. 1291-1300).

[54] Zhong, J., Guo, W., & Wang, Z. (2016, March). Study on network failure prediction based on alarm logs. In *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)* (pp. 1-7).

[55] Ji, W., Duan, S., Chen, R., Wang, S., & Ling, Q. (2018, June). A CNN-based network failure prediction method with logs. In *2018 Chinese Control And Decision Conference (CCDC)* (pp. 4087-4090).

[56] Javadi, A., Ganapathi, A., and Fox, A. (2019). FTA: A Large-Scale Fault-Injection Dataset for the Evaluation of Failure Prediction Methods. In Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 355-362.

[57] Harahap, E., Sakamoto, W., & Nishi, H. (2010, June). Failure prediction method for network management system by using Bayesian network and shared database. In *8th Asia-Pacific Symposium on Information and Telecommunication Technologies* (pp. 1-6).

[58] Xu, C., Wang, G., Liu, X., Guo, D., & Liu, T. Y. (2016). Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Transactions on Computers*, *65*(11), 3502-3508.

[59] Schroeder, B., & Gibson, G. A. (2009). A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing*, *7*(4), 337-350.

[60] Iosup, A., Jan, M., Sonmez, O., & Epema, D. H. (2007, September). On the dynamic resource availability in grids. In *2007 8th IEEE/ACM International Conference on Grid Computing* (pp. 26-33).

[61] Arpaci, R. H., Dusseau, A. C., Vahdat, A. M., Liu, L. T., Anderson, T. E., & Patterson, D. A. (1995). The interaction of parallel and sequential workloads on a network of workstations. *ACM SIGMETRICS Performance Evaluation Review*, *23*(1), 267-278.

[62] Bakkaloglu, M., Wylie, J. J., Wang, C., & Ganger, G. R. (2002). *On correlated failures in survivable storage systems*. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.

[63] Bhagwan, R., Savage, S., & Voelker, G. (2003). Understanding Availability. In *Proceedings of IPTPS'03.*

[64] Rood, B., & Lewis, M. J. (2007, September). Multi-state grid resource availability characterization. In *2007 8th IEEE/ACM International Conference on Grid Computing* (pp. 42-49).

[65] Saikat Guha, & Ravi Jain (2006). An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS).*

[66] Javadi, B., Kondo, D., Vincent, J. M., & Anderson, D. P. (2009, September). Mining for statistical models of availability in large-scale distributed systems: An empirical study of seti@ home. In *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems* (pp. 1-10).

[67] Javadi, B., Kondo, D., Iosup, A., & Epema, D. (2013). The Failure Trace Archive: Enabling the comparison of failure measurements and models of distributed systems. *Journal of Parallel and Distributed Computing*, *73*(8), 1208-1223.

[68] Arpaci, R. H., Dusseau, A. C., Vahdat, A. M., Liu, L. T., Anderson, T. E., & Patterson, D. A. (1995). The interaction of parallel and sequential workloads on a network of workstations. *ACM SIGMETRICS Performance Evaluation Review*, *23*(1), 267-278.

[69] Kondo, D., Taufer, M., Brooks, C. L., Casanova, H., & Chien, A. A. (2004, April). Characterizing and evaluating desktop grids: An empirical study. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* (p. 26).

[70] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, & Henri Casanova (2007). Characterizing resource availability in enterprise desktop grids. *Future Generation Comp. Syst., 23 (7), 888-903* .

[71] *sklearn.preprocessing.StandardScaler*. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[72] *SMOTE — Version 0.10.1*. (n.d.). SMOTE — Version 0.10.1. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

[73] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, "Data preprocessing for neural networks: a review," Neural Networks, vol. 14, no. 6, pp. 645-662, 2001. doi: 10.1016/S0893-6080(01)00041-8

[74] Zhang, Z., & Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems, 31*.

[75] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Curriculum Vitae

Name:	Chandrika Saha
Post-secondary Education and Degrees:	M.Sc. Candidate, Computer Science The University of Western Ontario 2021-2023 B.Sc., Computer Science and Engineering University of Barishal, Barishal, Bangladesh 2014-2019
Honours and Awards:	Western Graduate Research Scholarship (WGRS) The University of Western Ontario 2021-2022
Related Work Experience	Teaching Assistant The University of Western Ontario 2021-2023