UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Tek Raj Chhetri

Towards AI for cloud services reliability using combined metrics

Master's Thesis (30 ECTS)

Supervisor: Prof. Satish Narayana Srirama
Co-Supervisor: Chinmaya Dehury, PhD
Co-Supervisor: Artjom Lind, MSc

Tartu 2020

Towards AI for cloud services reliability using combined metrics

Abstract: With the emergence of cloud computing and the *Quality of Services (QoS)*, *Compute Power*, *Performance*, and *Scalability* it offers, the paradigm of computing has shifted towards the cloud. Due to attractiveness cloud offers, today, more and more businesses, research, and individuals are adopting cloud services. Even with the maturity of the cloud, reliability is still a concern. The reason being the constant occurrence of failure causes financial loss as well as a negative impact on its users as it directly affects QoS. Further, the scale and heterogeneity make it more prone to failure, highlighting the necessity for a robust solution to maintain the attractiveness and prevent financial loss. By predicting failure before it could happen, we can improve the reliability. Artificial Intelligence, now, has made significant progress, finding itself a place in all possible areas. In our study we present artificial intelligence with a combined metrics approach to improve the failure prediction. An experiment conducted with data recorded from more than 100 cloud servers shows significant improvement in failure prediction with high prediction accuracy, precision, and recall compared to state of the art studies.

Keywords: Cloud Computing, Artificial Intelligence, Failure Prediction, Reliability, Fault-tolerance

CERCS:P170 Computer science, numerical analysis, systems, control

Kombineeritud mõõdikute lähenemine pilve töökindluse säilitamiseks tehisintellekti abil

Lühikokkuvõte:

Pilvandmetöötluse ilmumisega ning sellega kaasneva teenuste kvaliteedi (QoS), arvutusvõimsuse, jõudluse ja mastaapsusega on arvutustehnika paradigma nihkunud pilve poole. Pilveteenuste atraktiivsuse tõttu üsna rohkem ettevõtteid, teadureid ja üksikuid on võtnud neid kasutuselle, vaatamata selle peale, et pilve usaldusväärsus on endiselt murettekitav. Pidevad läbikukkumised põhjustavad finantskahju ning mõjutavad negatiivselt kasutajaid, selle tulemiks on üdine QoS langus. Lisaks, pilve ulatus ja heterogeensus tõstab vastavalt tõrkekindlust, tuues esile huvi tugevama lahenduse vastu, selleks et, säilitada atraktiivsus ja vältida rahalist kahju. Prognoosides ebaõnnestumist ennem, kui see juhtuda võib, saavutame kõrgemat usaldusväärsust. Tehisintellekt on nüüd teinud märkimisväärseid edusamme, leides endale koha kõigis võimalikes valdkondades. Antud töö uurib tehisintellekti kombineeritud mõõdikute lähenemisviisi, selleks et parandada rikke ennustamist. Katse on viidud läbi enam kui 100 pilveserveri salvestatud andmetega ning näitab rikke prognoosimise olulist paranemist kõrge ennustusprotsessi täpsuse, tabavust ning tagasikutsumist võrreldes tehnika taseme uuringutega.

Võtmesõnad: pilvandmetöötlus, tehisintellekt, rikete ennustamine, usaldusväärsus, rikketolerants

CERCS:P170 Arvutiteadus, arvuline analüüs, süsteemid, juhtimine

Acknowledgements

I would first like to thank my supervisors Prof. Satish Narayana Srirama, Dr. Chinmaya Dehury, and Artjom Lind for providing direction to the correct path and their continuous support.

I would also like to thank the University of Tartu High-Performance Computing (HPC) center, especially Sander Kuusemets and Anders Martoja for helping me with access to the data.

I would also like to thank Professor Eero Vainikko who provided me an opportunity to work as a Teaching Assistant for the Parallel Computing course that help broadens my skills further and was very helpful while working on my thesis.

I would also like to thank Dr. Amnir Hadachi Intelligent Transportation Lab for providing an opportunity to be a part of the research activities in the Lab that helped to improve my technical and research skills.

I would also like to thank the University of Tartu and the Ministry of Foreign Affairs (MFA) Estonia for the scholarship that helps to make my study smooth. Finally, to the Archimedes Foundation for granting me with Kristjan Jaak scholarship for short term visits to attend Summer School "Enabling Technologies for Industrial IoT - 2019" during my studies.

Table of Contents

1 Introduction

Cloud computing is a model for delivering information services with the flexible use of virtual servers, massive scalability, and management services [1]. The popularity of cloud is due to increasing demand in diverse domains, cost-effectiveness with pay-as-you-go or subscription-based service model for on-demand access to IT resources [2], [3]. The other reason is the increase in productivity by reducing the number of chores required to be done by the IT teams [4]. It has been widely adopted by individuals, both private and public institutions and has emerged as the backbone of the modern economy [2]. Moreover, in recent years, new latency aware computing paradigm like fog computing, edge computing has evolved to meet real-time requirement [5]. Even with these computing paradigms like fog and edge computing, the cloud stays at the top of the hierarchy, highlighting the importance and new opportunities for the cloud.

Cloud today offers infrastructure support via IaaS (Infrastructure as a Service), a platform for software deployment via PaaS (Platform as a Service) and also software services via SaaS (Software as a Service). The application and importance of cloud computing today has grown remarkably. Cloud today is used to support smart city construction [6], smart manufacturing [7], enterprise business [8], scalable data analysis [9], [10], healthcare [11], [12]. The use-case of the cloud is increasing with the increase in technological advancement. Furthermore, it is forecasted to have around 500 Billion public cloud vendor revenue by 2026[1]. The major portion of this revenue goes to PaaS and IaaS 298.4 and 126 Billion respectively.

The attractiveness of the cloud is because of its reliability, low cost with pay-as-you-go, scalability, ease of programmability, high-performance dynamic resource provisioning [13, 14]. But with the scale, heterogeneity, and distributed nature of the cloud, failure is imminent [14], [15], [16]. The study has shown that a system with 100,000 processors experiences a failure every couple of minutes [16]. Though failure causes the system to breakdown, the distributed nature of the cloud, it only suffers from partial failure [14]. Such a failure can have a substantial financial loss and degradation of performance [13], [17].

Failure prediction is a mechanism that allows predicting the failure before it happens. It improves reliability. Higher reliability reduces loss like a financial loss. Therefore it is critical to have a failure prediction mechanism to avoid such a loss. Moreover, the study suggests that failure prediction is useful even if imperfect prediction and with limited precision [18]. For example, if only 50% is correctly predicted and remaining are incorrectly predicted, then it can at least save from 50% of the loss than not having anything.

The failure in the cloud can occur due to various reasons like hardware failure or

[1]Wikibon Research Cloud Computing (2015-2025) https://wikibon.com/wp-content/uploads/Wikibon-BGracely-Cloud-Computing-Nov-20152.pdf

software failure. Based on the fault-tolerance measure taken, it can be divided into two categories: Reactive Fault Tolerance and Proactive Fault Tolerance [19]. Figure 1 shows the categorization of fault tolerance technique based on steps taken to handle the fault. Reactive fault tolerance techniques are applied when failure has occurred while in proactive fault tolerance, earlier prediction of failure is made before it happens [19], [20]. The failure prediction, therefore, is a part of the proactive fault-tolerance technique.

Today, techniques and algorithms that have stemmed from the field of machine learning have indeed became a powerful tool for the analysis of complex and broad data, successfully assisting scientists in numerous breakthroughs of various areas of science and technology [21]. It allows computers to solve problems that before seemed to be unsolvable by computational processes alone[22]. It has resulted in state-of-the-art performance for many practical problems, especially in areas involving high-dimensional unstructured data such as computer vision, speech, and natural language processing [23]. Even with such advancement, still, the rule-based approach applying tools like Prometheus is used[2] for monitoring and earlier failure warning. This rule-based approach lacks robustness. It is not competent in detecting and learning patterns like artificial intelligence [24]. This article focuses on using artificial intelligence techniques to improve the reliability of cloud services by improving proactive fault-tolerance. Further, in the subsections of this chapter, we will discuss our motivation towards this work, goal and our contribution subsequently in subsections 1.1, 1.2 and 1.3.

1.1 Motivation

With the increase in the adoption of cloud computing more by businesses [47], there endures the challenge of providing scalable and reliable services. Efforts have been put into improving the performance, reliability, and scalability concern [48]. Reliability is one of the primary concerns and critical features in cloud computing, making a direct impact on QoS [16]. Therefore, in our study, we address the issue of reliability. Further due to the increase in ubiquitous computing the need for fault tolerance has increased giving rise to interest in the area of fault tolerance [25]. This concern of fault tolerance with the growth of ubiquitous computing is a motivating factor for us to find an approach to improve reliability making earlier failure prediction.

Several studies have been conducted in the area of failure prediction; however, a limited number of the conducted studies are focused on server failure prediction. It is our first motivation toward this study. The second motivation comes from the metric used. Earlier studies, as in Section 2, show the different parameters and their capability for use in failure prediction. However, the conducted previous studies fail to use the combined metrics that could improve the failure prediction. The combined metrics here mean the use of the hard drive attributes along with other metrics like CPU utilization, Memory

[2]https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

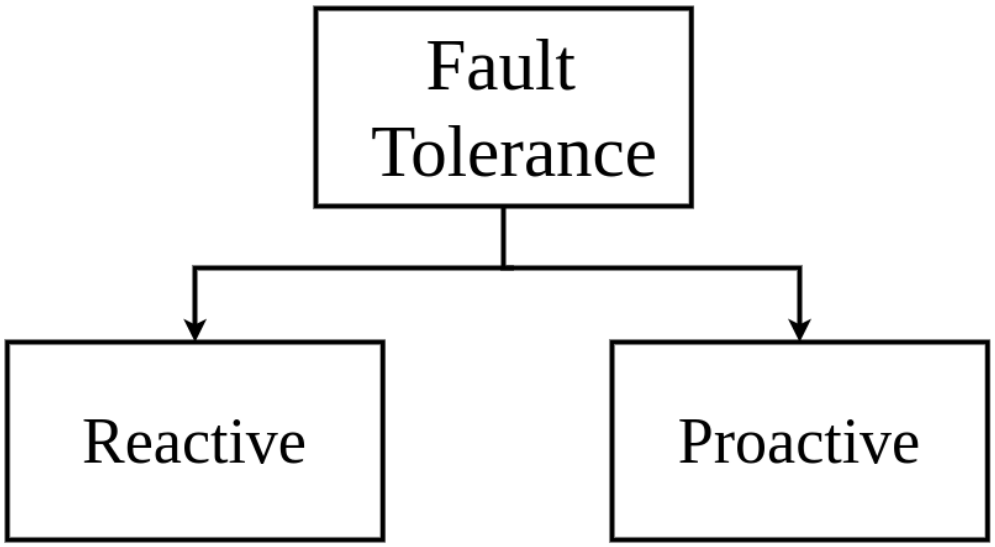

Figure 1. Classification of Fault-tolerance techniques

utilization, etc.

The artificial intelligence undoubtedly has made massive progress in recent years. It's capability of learning patterns from data makes it different from traditional approaches like a rule-based approach. Even with the advancement in artificial intelligence, a traditional rule-based tool like Prometheus [26] is still in use for monitoring failure. The third motivation is from the progress in the area of artificial intelligence in recent years. We, therefore, would make use of artificial intelligence taking advantage of its advancement to make failure prediction robust.

Finally, the availability of the data itself is a motivation for us.

1.2 Goal

Earlier failure prediction is vital to improve reliability because it allows us to take proactive action. The level of reliability, thus, depends on the accuracy of correctly predicting the failure. Due to this, there always has been emphasis put into improving the accuracy of correctly predicting failure. Therefore, the goal of this study is to improve reliability by improving the accuracy of correctly predicting failure with a combined metrics approach and artificial intelligence technique.

1.3 Contributions

To keep on maintaining the cloud popularity in the industry by providing reliable service and to save financial loss due to failure, this study proposes the robust failure prediction mechanism. The main contribution of this work can be summarized as follows:

- This study makes combined use of Self-Monitoring, Analysis and Reporting Technology (SMART) metrics along with other parameters like CPU utilization, Network Overhead, Memory utilization, Disk utilization to make failure prediction robust. SMART is used for monitoring the status of the hard drive.
- A data science approach using machine learning and deep learning is taken to overcome the disadvantage of rule-based failure prediction.
- A comprehensive evaluation of multiple techniques like RandomForestClassifier, GradientBoostingClassifier, LSTM (Long Short Term Memory), GRU (Gated Recurrent Unit) is performed.
- In contrast to some studies, this study makes use of real data from more than 100 cloud servers.

1.4 Outline

In this chapter, we had a brief introduction about cloud computing importance in the current era, the necessity of fault tolerance, and how artificial intelligence can help. In the next chapter that is Chapter 2 the discussion is focused on the background of fault tolerance and state-of-the-art in fault tolerance. In Chapter 3, we summarize our approach to the failure prediction problem. In this chapter, we presented the discussion on data collection and pre-processing, hyperparameter optimization techniques, algorithms used, activation, and loss functions. In a similar manner Chapter 4 summarizes how the experiment was conducted and discussion of result in Chapter 5. Finally, in Chapter 6 we provide the concluding statement and the future direction to this work.

2 State-of-the-art

In this section, we focus our discussion on related works done in the area of failure prediction in the cloud domain in Section 2.2. But before moving to our discussion to related works, we also have presented background discussion on Fault tolerance describing how the necessity of fault tolerance evolved and varied over time in Section 2.1.

2.1 Background

A fault is the manifestation of unexpected behavior, and fault tolerance is a mechanism that masks or restores the expected behavior of a system following the occurrence of faults [27]. In other words, fault tolerance is the ability of a system to continue performing its intended function despite faults [25]. Fault-tolerance is a mechanism that helps to maintain high reliability. Reliability which is the function of time t gives the probability that the system operates without failure in the interval [0, t], given that the system was performing correctly at time 0 [25]. Today the use of computers is everywhere like in a business, mission, and safety-critical applications. Depending on where it is implemented, the consequences can be devastating. For example, if we consider the safety-critical implementations of the computers, any occurrence of the fault involves loss of life. For the case of business, the occurrence of the fault makes a direct impact on the business as it involves cost and time.

From the earlier days, failure always has been a concern. People were taking small measures for handling the occurrence of the fault. The focus was very less in the direction of fault tolerance. It is World War II after which fault tolerance gained momentum [25]. The use of the electronic equipment in war and the occurrence of failure gave the realization for the need for fail-safe systems. Further was fueled by the "space race" leading to the initiation of different projects related to fault tolerance [25].

Similarly, from the beginning of the distributed computing (DC) era that occurred around the mid-1970s, fault tolerance has been touted as one of the main advantages of shifting the computer system design from the centralized structure into a more distributed structure [28]. This is because of the nature of the distributed systems. Distributed systems consist of multiple connected computers (systems) coordinating with each other. Because of their distributed nature, they do not suffer complete failure like a centralized system. Suppose a non-distributed system, any occurrence of the failure affects the whole system. But this is not the case for a distributed system because the distributed system is not a single computer rather a collection of computers connected via network. Though the system does not experience complete shutdown, it, however, is affected in terms of capacity and performance.

In recent years, the interest in fault tolerance has been further boosted by the ongoing shift from the traditional desk-top information processing paradigm, in which a single

user engages a single device for a specialized purpose, to ubiquitous computing in which many small, inexpensive networked processing devices are engaged simultaneously and distributed at all scales throughout everyday life [25]. Cloud computing is one of them. It is a distributed computing that allows the pay-per-go model for use of services like infrastructure, platform. Today cloud computing has support for different sectors like business, research, academia, and has emerged as a backbone of the economy [2]. Even though the distributed nature of the cloud, it does not occur complete failure but the constant occurrence of the failure is not desirable as the loss involves money. Moreover, the layered architecture of the cloud failure in one layer impacts the above layers.

There are different strategies used based on the type of fault in the cloud computing environment. The techniques like self-healing, preemptive migration, system rejuvenation for proactive fault tolerance technique and check-pointing, replication, and job migration for reactive case [29]. However, there has been a change in dealing with fault and new research direction with artificial intelligence is being explored as described in Section 2. Further, a rule-based tools like Prometheus[4], Nagios[5] are used for monitoring the cloud infrastructures.

2.2 Related Work

From the background, we have seen how the need for fault tolerance evolved and changed over-time. In this section, we present a brief survey of recent works in failure prediction in the domain of cloud computing.

2.2.1 Cloud VM Failure Prediction

Meenakumari et al. [30] use the dynamic threshold approach for earlier VM failure prediction using CPU utilization, CPU usage, bandwidth, temperature and memory as metrics. This study poses advantages over earlier studies that take a proactive approach which requires failure to be predictable. This study is not using the machine learning approach as our study.

Alkasem et al. [31] in his study present a case study of Virtual Machine (VM) failure to start-up using machine learning (Naive Bayes Classifier) and Apache Spark Streaming. The metrics CPU Utilization, Memory Usage, Network Overhead, IO Storage usage is considered for this study. This study, though, makes use of the machine learning approach but fails to use the SMART, hard drive metrics in a combined manner like our study.

Qasem et. al. [32] uses A Fuzzy min-max Neural Network classification approach is proposed to predict virtual machine failure. This study uses CPU utilization, memory utilization, job capacity, and CPU temperature resource utilization metrics. Along with Fuzzy min-max Neural Network KNN is used in case of overlapped hyper box, to classify overlapping. The study, however, is conducted with simulated data from cloudsim. Liu

et. al. [33] uses Recurrent Neural Network model to improve the reactive hard-drive fault-tolerance techniques for the cloud storage systems using SMART attributes.

2.2.2 Server Failure Prediction

Mohammed et al. [34] in his study applied multiple machine learning algorithms KNN, SVM, RF, CART, LDA and evaluated them on accuracy of predicting HPC system failure. In this study, the authors consider the following sources of failure; Hardware, Software, Human Error, Network, and Undetermined [35]. Unlike our study, this study takes human error and other undetermined errors into account. Using such, we cannot say if the system failed actually or there was human intervention.

Xu et. al. [36] predicted the disk error using a ranking based machine learning approach. The early prediction allowed migration of VM to a healthy node thereby improving the service availability of Microsoft Azure. However, this study is limited to only using disk information.

Lai et al. [37] in his study predict whether a server fails within 60 days using hard drive data collected from SLAC Accelerator Laboratory and maintained failure logs over 10 years. The authors also introduced *time_since_prev_failure* that represents the time since the previous failure and KNN and 16 days sliding window-based KNN for making a prediction. This study is only based on hard disk data unlike ours.

Das et. al. [18] in his study uses LSTM to predict the failing node so that the computation from the failure node could be migrated to the healthy nodes. The author makes use of the data from Cray XC30, Cray XE6, Cray XC40, Cray XC40/XC30. This study, unlike other studies, uses the error phrases from the system log. While Tehrani et. al. [38] using SVM and metrics temperature, CPU, RAM, and bandwidth utilization make failure prediction. The study clearly lacks the benefit of other attributes like hard drive information as in our study.

The study by Chigurupati et al. [39] focuses on predicting hardware failure combining the machine learning approach using communication failure information, predicting failure 5 minutes ahead. The study is limited to using only communication failure.

Ganguly et al. [40] predict the failure in cloud systems using a two-stage ensemble model, Decision Tree, and Logistic Regression, using SMART data and windows performance counter[3]. Similar to it, Lima et al. [41] predicts hard drive failure using LSTM. This study only considers hard disk failure predictions.

Using Linear Regression and SVM with Gaussian kernel, Adamu et al. [42] studied and analyzed NERSC data collected from CFDR. The performance evaluation is performed using K-folds cross-validation. The author presents the prediction of Disk, DIMM (dual in-line memory module), CPU, and Other failures separately in the bar graph representation. It is not clear what is the scope of other failures and no network

[3]https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal

information is used as it is also one of the reasons for failure.

2.2.3 Task Failure Prediction

Shetty et al. [43] conducted a study using the Google cluster dataset for resource usage by the task and failure prediction using the XGboost classifier. Similarly, Jassas et al. [44] also uses the Google cluster trace data for analyzing job failure in the cloud computing environment. As opposed to our study, these studies are focusing on the job or task failure.

Bala et al. [45] conducted a study for proactive fault tolerance by predicting the task failures for Scientific Workflow applications using the machine learning approach. This study makes use of algorithms like Naive Bayes, Random Forest, Logistic Regression, and Artificial Neural Networks. This study though uses the machine learning approach for proactive failure prediction but is limited to task failure prediction.

Rosa et al.[46] conducted studies about job failure prediction using metrics like CPU utilization, Memory utilization, Disk utilization, and data from Google cluster traces. It makes use of y Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Logistic Regression (LR) approach. Similar to other task failure studies, it is also focused on the failure of the task only.

Gao et al. [47] studies a multi-layer Bidirectional Long Short Term Memory (Bi-LSTM) approach to identify tasks and job failures in the cloud. Gao et al. also makes use of Google cluster trace data for his study. This study makes use of the deep learning approach but is focused on task and job failure.

2.3 Summary

In this chapter, we first introduced with background information on fault tolerance. Then we discussed the current state-of-the-art in failure prediction in the cloud computing domain. We also saw the different approaches used to answer the failure prediction issue in related works 2.2. Finally, we saw what is new in our studies compared to the current state-of-the-art. In the next chapter, we shall see our approach to the problem of failure prediction.

3 Methodology

For a problem, multiple approaches are leading to achieve the same solution. In this section, we describe our path to solve the problem of predicting failure. The discussion involves details about data collection and selected algorithms. We first begin our discussion with system architecture in Section 3.1 and proceed with how we collected the data, need to collect data in Section 3.2, thereby moving our discussion to data processing in Section 3.3. In Section 3.4 we discuss about hyperparameter optimization and its requirements in detail. Section 3.5 involves the detailed discussion of our first algorithm that we have selected for our study and Section 3.6 for another algorithm Gradient Boosting. We then move forward with our discussion of the deep learning-based methods that we have used in our study. The deep-learning methods we have selected for our study are based on RNN, hence Section 3.7 is dedicated to the discussion of the Recurrent Neural Network. Then in Section 3.8, we describe the first deep-learning-based method Long Short Term Memory (LSTM) that we have used in our study and the other method called Gated Recurrent Unit (GRU) in Section 3.9. Finally, we conclude this section with the discussion of the Activation Functions in Section 3.10.

3.1 Architecture

Figure 2 shows the architecture of the proposed work. We first begin with data collection. The data collection involves downloading the data that is monitored by Prometheus[4]. Prometheus is an open-source monitoring and alerting toolkit backed by a rule-based engine [26]. The downloaded data is then converted into CSV format which acts as input data. The preprocessed input CSV data is then split into train and test data. Using the test data, we train our model with the selected algorithm: Random Forest, Gradient Boosting, Long Short Term Memory (LSTM), and Gated Recurrent Unit (GRU). We then evaluate our model with the test data. The process of training and testing is performed separately for each algorithm. The evaluation of the model is performed with evaluation metrics like Accuracy, Precision, Recall, and F1 Score. Finally, the evaluated result from each of the algorithm is then compared among the proposed method as well as with other state of the art methods.

[4] https://prometheus.io/docs/introduction/overview/

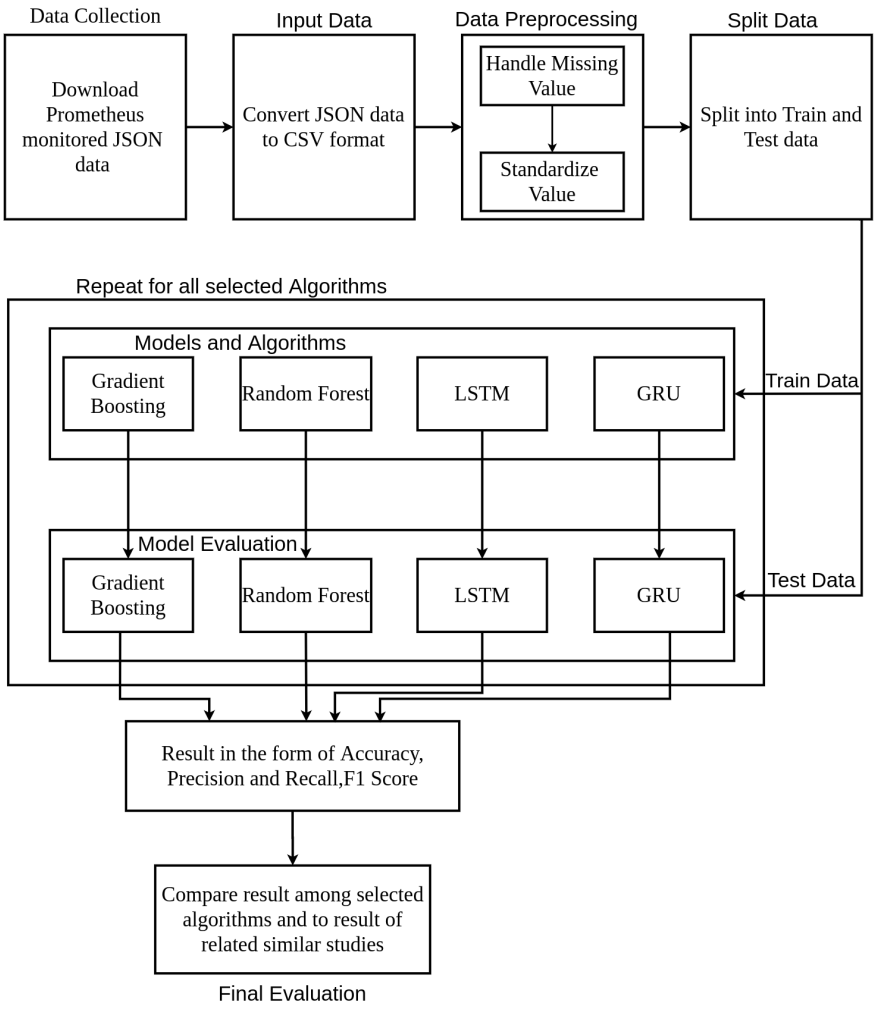

Figure 2. Architecture

3.2 Data Collection

Today large IT distributed infrastructure use different tools for monitoring. Some of the tools available today for monitoring are Nagios[5], Prometheus[4], Ganglia[6], Collected[7]. The purpose of using monitoring tools like Prometheus, Nagios is to know the available

[5]https://www.nagios.org/
[6]https://developer.nvidia.com/ganglia-monitoring-system
[7]https://collectd.org/

status of the monitored system like how much load system is currently having and further to support operations like alerting, debugging [48]. The cloud infrastructures that we consider for our study are monitored using monitoring tool Prometheus[4]. This monitoring provides the state of the technological environment [26]. This valuable information that is logged by the Prometheus can be used for improving proactive fault-tolerance by leveraging artificial intelligence techniques.

An artificial intelligence approach assumes learning the patterns from the data, therefore data is essential in any artificial intelligence approach. Without data, it is impossible to apply these artificial intelligence-based techniques to solve real-world problems. Therefore the data collection forms an essential component of such approach. Moreover, the quality of data has a direct impact on the accuracy of the result [49]. For example, if we consider medical areas like radiology, clinical trial [50], where today, machine learning approaches are used then one could only imagine the disaster that would be caused by the use of poorly collected data. The quality of data can only be achieved with a good understanding of the tools used for monitoring, the monitored environment, and the attributes to be collected. For example, let us consider the CPU utilization and smart metric SMART_198. Now if we just have the value 70 for CPU utilization and 5 for SMART_198 we cannot say how good the data is. The reason is that we are unaware of how and what tools are used for data collection. Further, the information about the collected attribute is not provided. We can determine the quality of data only when we have information about the collected attribute, used tools, and the monitored environment.

The failure dataset available is minimal. Among the available dataset, most of the data are related to job failure like Google Cluster[8]. While the other available data SMART[9] mostly contains the hard disk related failure. The lack of availability of data as per our requirement, we are required to collect the data by ourselves.

Prometheus logs a wide range of metrics. But not all recorded parameters can be used in every study. The selection of the metrics should be made based on its suitability, according to the study. It requires a review of the available metrics and their impact. Based on the detailed study of the available metrics and in-depth analysis of their effect on the system, careful selection of the metrics shown in Table 1 is made. The choice of the metrics CPU Utilization, Memory Utilization, Network Overhead, IO Utilization, Bits Read, Bits Write is based on its proven capacity in failure prediction from existing research like [30], [31], [32], [37], [38]. Similarly, SMART metrics are based on comparable such studies as [40], [40], [41]. Further, Mashhadi et al. [51] provide a detailed description of different SMART attributes.

[8]https://github.com/google/cluster-data

[9]https://www.backblaze.com/b2/hard-drive-test-data.htmldownloading-the-raw-hard-drive-test-data

SN	Metrics	Description
1	CPU Utilization	Host CPU Usage in %.
2	Memory Utilization	Memory Usage in bytes
3	Network Overhead	Network Usage in bytes
4	IO Utilization	IO Usage in time
5	Bits Read	Data written out from disk in bytes
6	Bits Write	Data written into disk in bytes
7	Smart 188	Command Time out
8	Smart 197	Current Pending Sector Count
9	Smart 198	Uncorrectable Sector Count
10	Smart 9	Power-On hours
11	Smart 1	Read Error Rate
12	Smart 5	Reallocated Sectors Count
13	Smart 187	Reported Uncorrectable Errors
14	Smart 7	Seek Error Rate
15	Smart 3	Spin Up Time
16	Smart 4	Start/Stop Count
17	Smart 194	Temperature
18	Smart 199	UltraDMA CRC Error Count

Table 1. Selected Metrics for the Study

The CPU utilization in our study represents how much the host CPU is utilized represented in percentage. In other terms, it represents how much load CPU is having. It is calculated on per node based on the current load on each CPU. For example, if there is A CPU load on a node which is having B cores then it is calculated as $(A/B) \times 100$. However, memory utilization and network overhead both represented in bytes gives information about how much memory is used in bytes and how much network usage is. The memory utilization likewise is calculated using the total available memory and available free memory information and the network overhead using information receive and transmitted over the network. In calculating the memory utilization, we do the difference operation between total memory and available free memory. While in the case of the network overhead, we sum both receive and transmit information. The remaining metrics can be obtained using Prometheus without needing to do any calculation. The IO utilization also referred as disk IO utilization gives information on utilization of the disk for IO operations and this utilization is calculated based on IO time in seconds. The other metric Bits Read and Bits Write tells us how much data is going in and out of the disk. The smart metrics tell us the status of the hard drive represented by value range 1 to 253. But most of the vendor uses the normalized representation, and it was the case

for us. The detailed description of the used SMART attributes are given below [10,11,12,13].

- **Smart 188**: This attribute is also called Command Time out and it represents the interrupted or aborted operations numbers due to disk timeout.
- **Smart 197**: This SMART attribute Smart 197 or Current Pending Sector Count provides the information about the unstable pending sector waiting to be remapped to spare area.
- **Smart 198**: It is also called Uncorrectable Sector Count or Off-line Uncorrectable Sector Count. It gives the information about Uncorrectable error count detected during the SMART scan. These are the errors that occurred when reading/writing.
- **Smart 9**: This Power-On hours SMART attribute gives the total time the drive is powered on or lifetime hours drive is power on.
- **Smart 1**: It gives the information about the error occurred while reading data from disk. It is also called Read Error Rate.
- **Smart 5**: This SMART attribute Reallocated Sectors Count gives information about a number of retired blocks since leaving the factory or grown defect count.
- **Smart 187**: It is also called Reported Uncorrectable Errors which gives information about the number of errors that could not be corrected using hardware error correction codes (ECC).
- **Smart 7**: This SMART attribute is also called Seek Error Rate. It measures the error that occurred during the positioning of the disk head while reading/writing.
- **Smart 3**: This SMART attribute is also called Spin-Up Time. It is a time needed by spindle to spin-up to full revolutions per minute (RPM). RPM measures how many times that disc spins in a minute.
- **Smart 4**: This Start/Stop Count or SMART 4 provides information about the estimated remaining life, based on the number of spin-up/spin-down cycles.

[10]Intel Solid-State Drive Data Center for SATA SMART Attributes https://www.intel.com/content/dam/support/us/en/documents/solid-state-drives/Intel_SSD_Smart_Attrib_for_SATA.PDF

[11]List of S.M.A.R.T. attributes http://www.cropel.com/library/smart-attribute-list.aspx

[12]TN-FD-40: 5200 SSD SMART Implementation Introduction https://www.micron.com/-/media/documents/products/technical-note/solid-state-storage/tnfd40_5200_ssd_smart_implementation.pdf

[13]Hard Disk Sentinel Help - S.M.A.R.T. attribute list https://www.hdsentinel.com/help/en/56_attrib.html

- **Smart 199**: It is also called UltraDMA CRC Error Count and it gives Frame Information Structure (FIS) cyclic redundancy check (CRC) error.
- **Smart 194**: It gives information about disk temperature.

Once everything is final, the next task is to download the data. The data downloading process of monitored infrastructure by Prometheus can be represented by Figure 3. The downloaded data is saved into the target format for data-science, tabular form [24] as a CSV file. As shown in Figure 3, we use the python script to download the data from Prometheus. Prometheus allows the query of the metrics logged by it via HTTP API[14]. The python script responsible for downloading the data makes use of HTTP API to query the data from Prometheus. For each query made Prometheus returns the data in a JSON shown in Figure 4. In the sample data, Figure 4, to anonymize data the details cluster, department, instance original information is not present. Though Prometheus allows the query of the metrics collectively, the query is made individually. It is done to avoid the failure that occurs due to timeout and Prometheus maximum resolution of 11000 points per time-series. The individual result of the query is combined based on timestamp. For the easiness, the data header is formatted according to the metrics used as in Table 1. Finally, the data is saved in the CSV format.

Prometheus is used for real-time monitoring of infrastructures to alert based on the rules specified. It also allows the data retention up to the duration as defined in the settings. It was 30 days for our case. It means we can have access to only 30 days of data. For our study, therefore, we recorded one-month data within an interval of 30 minutes. The recorded data is if size 986.2MB.

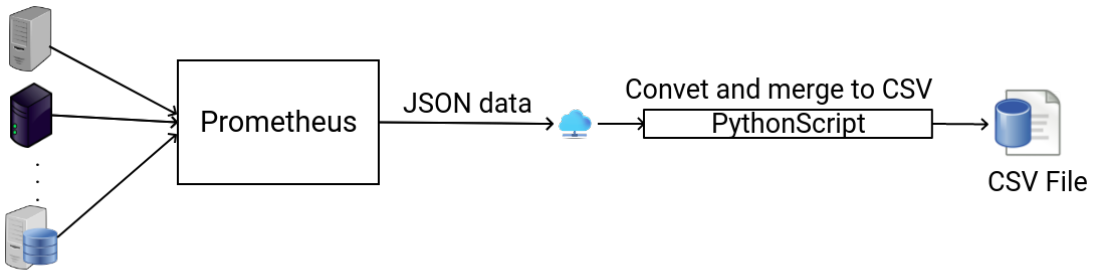

Figure 3. Prometheus Monitoring and Data Downloading

[14]https://prometheus.io/docs/prometheus/latest/querying/api/

```
{"status":"success","data":{"resultType":"matrix","result":[{"metric":
{"cluster":"XXX","department":"XXX","instance":"XXX","job":"metrics","role":"XXX"},
"values":[
[1583777451.781,"29363990528"],[1583779251.781,"29476913152"],[1583781051.781,"29335060480"],
[1583782851.781,"28781330432"],[1583784651.781,"28739928064"],[1583786451.781,"28728094720"],
[1583788251.781,"28736397312"],[1583790051.781,"28728963072"],[1583791851.781,"28731764736"],
[1583793651.781,"28726280192"],[1583795451.781,"28727107584"],[1583797251.781,"28730277888"],
[1583799051.781,"28732407808"],[1583800851.781,"28735016960"],[1583802651.781,"28740685824"],
[1583804451.781,"28729356288"],[1583806251.781,"28734312448"],[1583808051.781,"28742156288"],
[1583809851.781,"28733386752"],[1583811651.781,"28740583424"],[1583813451.781,"28733878272"],
[1583815251.781,"28738060288"],[1583817051.781,"28726018048"],[1583818851.781,"28729917440"],
[1583820651.781,"28732305408"],[1583822451.781,"28732071936"],[1583824251.781,"28738183168"],
[1583826051.781,"28741193728"],[1583827851.781,"28742692864"],[1583829651.781,"28736487424"],
[1583831451.781,"28858527744"],[1583833251.781,"56928935936"],[1583835051.781,"28871454720"],
[1583836851.781,"28876230656"],[1583838651.781,"34174541824"],[1583840451.781,"28923068416"],
[1583842251.781,"34940235776"],[1583844051.781,"28864970752"],[1583845851.781,"31437119488"],
[1583847651.781,"32037072896"],[1583849451.781,"32360796160"],[1583851251.781,"32458723328"],
[1583853051.781,"32696397824"],[1583854851.781,"32904458240"],[1583856651.781,"33053474816"],
[1583858451.781,"32699207680"],[1583860251.781,"41448972288"],[1583862051.781,"32097726464"],
[1583863851.781,"33784139776"],[1583865651.781,"33825603584"],[1583867451.781,"34025594880"],
[1583869251.781,"33129558016"],[1583871051.781,"33182605312"],[1583872851.781,"34303115264"],
[1583874651.781,"33451704320"],[1583876451.781,"33598652416"],[1583878251.781,"34645729280"],
[1583880051.781,"34823147520"],[1583881851.781,"32613269504"],[1583883651.781,"35052773376"],
[1583885451.781,"33986199552"],[1583887251.781,"35138641920"],
```


Figure 4. Sample Data of Prometheus Query

3.3 Preprocessing

The real-world data is not clean; it is incomplete, inconsistent, and noisy [52]. The data directly obtained from the real-world is, therefore, of not the quality. The quality of data, however, has a direct impact on the result. So, it becomes necessary to preprocess the data. The main objective of the preprocessing is to improve the quality of the data. Therefore improving the quality of data, we are indirectly contributing to the more desirable result. It involves performing one of the following tasks; Data Cleaning, Data Integration, Data transformation, Data reduction, Data discretization [43].

To make the data following the need for our study and improve quality, we also perform the data preprocessing. Some preprocessing, like target label generation, is performed during the time data is downloaded. Since we are doing supervised learning, we require a target label to train our model. However, the original data lacks this target label. So we use Target label generator algorithm 1 to generate the required target label for our study. Similar approaches have been followed in other studies like [53]. The failure characteristics from the studies like [18], [44] show high resource consumption. Moreover, the hardware comes with the device criteria that might lead to potential failure[15,16]. We use this information to define the failure criteria using Algorithm 1. For example, let us consider the metrics CPU utilization, smart_198. Studies like [18], [44] have shown a high correlation with the occurrence of failure and resource consumption.

[15]https://wiki.unraid.net/Understanding_SMART_Reports

[16]https://www.backblaze.com/blog/hard-drive-smart-stats/

Further, the hardware manufacturer also defines the breaking point for the particular hardware. If we consider the case of the hard disk then we can obtain this information from the SMART metrics. Now, say we set CPU utilization value to 100 and smart_198 to 30 as threshold value than providing this information in Algorithm 1 we can get the target label that tells us whether it is a failure or not. However, unlike this example, we in our study pass similar information about all the selected metrics shown in Table 1 to the Algorithm 1 to generate the target label.

Like any other data which generally consists of missing information, our data also contain a missing value. The missing values are incompatible with working. To make it compatible working with and fixing the missing value, we use a scikit-learn package. Scikit-Learn is a machine learning library in Python that provides implementations of classification, Regression, Clustering, Preprocessing, dimensionality reduction, and Model Selection algorithms[17]. One strategy to work with missing values it to discard the row or column with missing values. However, in doing so, we might lose the meaningful information. For example, a row contains a missing value. If we use the deletion strategy, then we should delete the entire row. For one missing value, we are removing the entire row. It is not ideal as we see that we are losing the other useful information present in that row. To avoid such a situation, we use the Scikit-Learn Imputation technique SimpleImputer. The SimpleImputer provides mean, median, most frequent, constant imputation strategy[18]. We use the mean imputation strategy, which is also the default strategy[18]. The value in our data is not uniform. Different metrics have different value range. It creates a problem, and our model tends to be biased. To deal with this issue and make all the values in the same range, we use Scikit-Learn StandardScalar. It Standardizes features by removing the mean and scaling to unit variance[19].

3.4 Hyperparameter Optimization

By default, the machine learning algorithms come with a predefined value of hyperparameters. It can be considered as input configurations for the algorithm. This preset value of hyperparameters is not suitable for all the tasks. So, we change the value of these hyperparameters based on our experiment to improve the accuracy of the result. But it is complicated because it is often a nonintuitive, time-consuming, and systematic trial-and-error process [54]. Further, difficulty is exacerbated because hyperparameters must be set manually before training can even begin and expertise in parameter tuning can only be achieved from experience with data and data sets, diligence, hard work, and just plain practice [54]. We, therefore, use the hyperparameter optimization techniques to find the suitable hyperparameter configuration.

[17]https://scikit-learn.org/stable/index.html

[18]https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

[19]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Algorithm 1: Target Label Generator algorithm

Data: ipData ← input data without target label in dictionary format
Result: Data with target label

```
data ← ipData;
targetValue ← Initialize with an empty List;
resultdata ← Copy of data;
N ← len(data);
for i=0 to N do
    whatLabel = [cpu_utilization[i] > CPU_UTILIZATION_THRESHOLD,
     memory_utilization[i] > MEMORY_UTILIZATION_THRESHOLD,
     network_overhead[i] > NETWORK_OVERHEAD_THRESHOLD,
     io_utilization[i] > IO_UTILIZATION_THRESHOLD, bits_outputted[i] >
     BITSOUTPUTTED_THRESHOLD, bits_inputted[i] >
     BITSINPUTTED_THRESHOLD, smart_188[i] <
     SMART188_THRESHOLD, smart_197[i] < SMART197_THRESHOLD,
     smart_198[i] < SMART198_THRESHOLD, smart_9[i] <
     SMART9_THRESHOLD, smart_1[i] < SMART1_THRESHOLD,
     smart_5[i] < SMART5_THRESHOLD, smart_187[i] <
     SMART187_THRESHOLD, smart_7[i] < SMART7_THRESHOLD,
     smart_3[i] < SMART3_THRESHOLD, smart_4[i] <
     SMART4_THRESHOLD, smart_194[i] > SMART194_THRESHOLD,
     smart_199[i] < SMART199_THRESHOLD ];
    if True in whatLabel then
        AppendTotargetValue(1)
    else
        AppendTotargetValue(0)
    end
end
resultdata ← add targetValue to new target column;
return resultdata;
```

Let **A** be any machine learning or deep learning algorithm with hyperparameters λ_1,λ_2,...,λ_n. Then the hyperparameter optimization (or search) is the process of finding the optimal value of hyperparameters that yields better result by minimizing the validation loss [55], [56] [57]. Under k-fold cross-validation, with training data D_{train} and validation data D_{test} the hyperparameter optimization for learning algorithm A with hyperparameter λ, represented by A_λ, minimizes the validation loss as Equation 1 [55].

$$f(\lambda) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_\lambda, D^i_{train}, D^i_{test}) \tag{1}$$

where,
$f(\lambda)$ represents the validation loss minimizing function

K represents the number of cross-validation folds

$\mathcal{L}(A_\lambda, D^i_{train}, D^i_{test})$ represents validation loss

The hyperparameters can be an integer or a categorical value. It depends on the algorithm. Furthermore, the hyperparameter of the two algorithms need not necessarily be the same, though sometimes there can be common attributes. There are different techniques for hyperparameter optimization like Grid Search, Random Search, Bayesian optimization [57].

Moreover, today, machine learning itself and hyperparameter optimization is moving towards automatic process [55] [58] [59] with the availability of the frameworks Optuna [60]. Similarly, Scikit-Learn also provides a library for automatic hyperparameter configuration [61], [62]. Despite it, we have selected Grid Search as a choice for hyperparameter optimization. It is because of the simplicity that Grid Search offers and is also the reason for it prevailing as state of the art despite decades of research into global optimization [63].

GridSearchCV is one of the hyperparameter tuning (optimizing) methods that Scikit-Learn offers. It performs an exhaustive search over the specified parameters grid along with cross-validation. For each combination, it conducts the cross-validation and selects the one with the highest accuracy as the final result for the model. The working of the GridSearchCV is shown in Figure 6. The GridSearchCV consists of the following key components [64]. The official documentation[20] provides detailed information on all available parameters.

- **Search grid:** The search grid is also be referred to as the parameter grid. It is a dictionary representation of the set of hyperparameters. The combination of

[20]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

these hyperparameters will be used in GridSearchCV to find the best one that will maximize the performance of the model. Figure 5 shows an example Random Forest search grid.

```
{
'n_estimators':[10,20,60],
'max_features':['auto','log2','sqrt'],
'criterion':['gini','entropy'],
'min_samples_split': [3,5,8]
}
```

Figure 5. Example Search Grid

- **Estimators:** It is used to implement the Scikit-Learn estimator. The estimators are the machine learning models we are trying to optimize. It could be a classifier or regressor.
- **Cross-validation:** Cross-Validation which belongs to the family of Monte Carlo methods is a data resampling method to assess the generalization ability of predictive models and to prevent overfitting [65]. It evaluates and compares the learning algorithm by dividing the data into two segments; training and validation data [66]. Figure 7 shows the K-Fold cross-validation. The data is divided into k disjoint folds of approximately equal size [65], [67]. The fold here means subsets. One important thing about K-Fold cross-validation is that no two test sets overlap [65]. Each fold is then used once as validation and remaining k-1 fold as the training by the machine learning model [67]. The accuracy is then calculated by averaging the accuracy of the K folds.

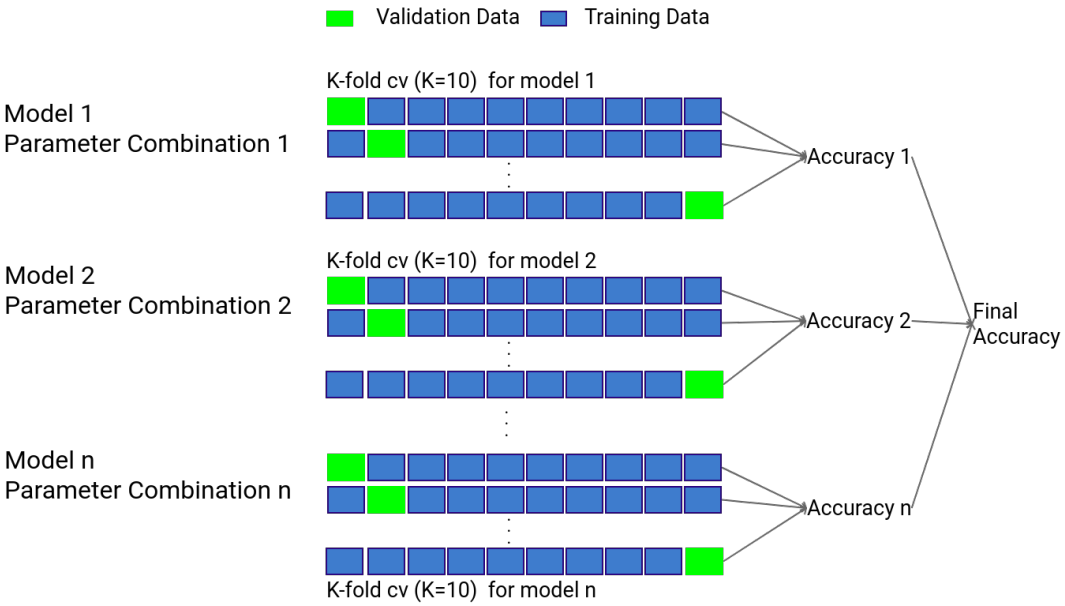

Figure 6. GridSearchCV

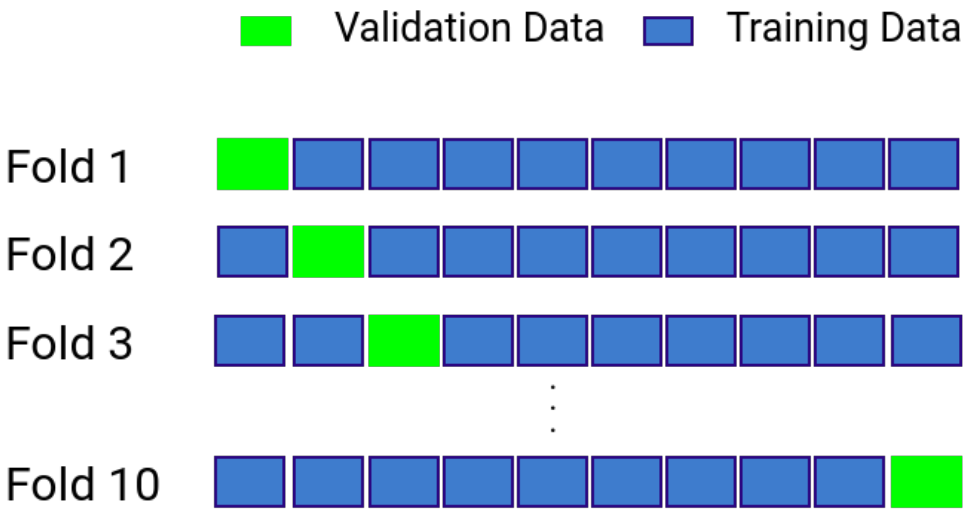

Figure 7. K Fold cross-validation (K=10)

In our study, to make the suitable selection of the hyperparameters, we also make use of the hyperparameter optimization. The Scikit-Learn Grid Search CV was used for the hyperparameter optimization in our study. It performs the Grid Search as well as the K-Fold cross-validation. For our experiment, we use the three K-Fold cross-validations with the Grid Search. Table 2 shows Random Forest Grid Search search grid for hyperparameter optimization used in our study. Similarly, Table 3 shows the Grid Search search grid used in our study for hyperparameter optimization for the Gradient boosting algorithm.

Parameter Name	**Value**
n_estimators	[100,200,300,500]
max_features	['auto','log2','sqrt']
criterion	['gini','entropy']
min_samples_split	[3,5,8,9,10,30,50]

Table 2. Random Forest Hyperparameter

Parameter Name	**Value**
loss	['exponential','deviance']
learning_rate	[0.001,0.01,0.0001]
max_features	[2, 3,5,7]
min_samples_split	[3, 4, 5,7,9]
n_estimators	[100,200,300,500]

Table 3. Gradient Boosting Hyperparameter

3.5 Random Forest

Of the many available machine learning algorithms, the random forest has outstanding practical performance and is widely used [68]. Because of its robustness, it is used in different application areas like for Classification of Neuroimaging Data in Alzheimer's Disease [69], Land Cover Classification [70], Stock Prediction [71], Remote sensing like Global Burned Area Classification [72], Assessment of Carbon Stocks [73]. Because

of the advantages it posses, we in our study have selected it as one of the methods for failure prediction.

Random Forest is a classifier consisting of a collection of tree structured classifiers $\{h(x, \theta_k), k = 1, ...\}$ where the $\{\theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x [74]. It is an ensemble learning method for classification and regression. It makes use of the randomized decision tree during the training. It is similar to the bagged decision tree; however, it differs in feature use while learning in each recursive step of decision trees described as follows[21].

- Randomly select F features out of all P given features.
- Find the best split among these features.

The value of F varies for regression and classification. It is $\frac{P}{3}$ in case of regression and $\sqrt{(P)}$ for classification [68]. The best split is selected by optimizing the Classification and Regression Trees (CART) split criterion, based on the so-called *Gini impurity* (for classification) or the prediction squared error (for regression) [68], [75]. As we are performing the classification task, we will limit our discussion on classification. Gini impurity measures how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset [22]. While entropy is the expected number of bits needed to encode a randomly drawn value of random variable [23]. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node[24]. By default, Scikit-Learn uses Gini as the split measure. It also, however, provides an option to choose Entropy as the split measure. Information gain is calculated using either Entropy or Gini based on the split measure used. Depending on the value of Information Gain, the split of the tree is performed. The Gini Impurity[25], Entropy[25] can be calculated following the Equation 2 and Equation 3 respectively. Similarly, Equation 4[21] can be used to calculate Information gain for Gini based split and Equation 5[21] for Entropy-based split.

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \tag{2}$$

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{3}$$

[21]Ensemble methods, Meelis Kull, University of Tartu `https://drive.google.com/file/d/1vgXQcperDN8dqKYIrmV0kH3VDhHXs-pK/view`

[22]https://www.cs.purdue.edu/homes/ninghui/courses/Fall18/handouts/07_dtree.pdf

[23]https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture11.pdf

[24]https://scikit-learn.org/stable/modules/tree.htmltree

[25]https://cs.anu.edu.au/courses/comp3420/mining/data-mining-05.pdf

$$I_G = Gini_p - \frac{SizeT_L \times GiniT_L + SizeT_R \times GiniT_R}{Size_p} \tag{4}$$

$$I_G = Entropy_p - \frac{SizeT_L \times EntropyT_L + SizeT_R \times EntropyT_R}{Size_p} \tag{5}$$

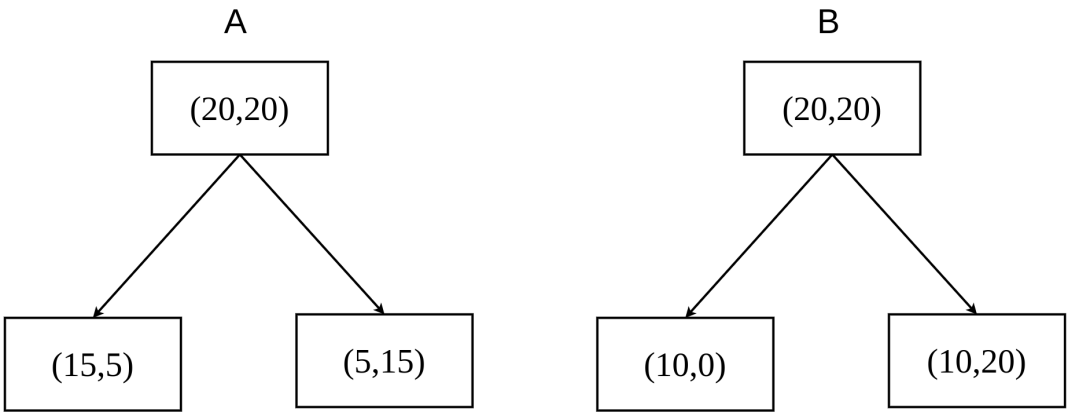

Figure 8. Sample Tree

For example, if we consider the tree shown in Figure 8, then selecting the best split is decided based on the information gain value, whichever the split metric Gini or Entropy is chosen. Table 4 shows the Information Gain calculation for Tree A using both Gini and Entropy. Likewise, Table 5 shows the calculation of Information Gain for Tree B. Based on the calculated information gain value, Random Forest, would choose Tree B for the split as it has got higher information gain for both Gini and Entropy. We have shown both Gini and Entropy-based calculations. But in the implementation, either one of them is used.

Gini	Entropy
$Gini_p = 1 - ((\frac{20}{40})^2 + (\frac{20}{40})^2) = 0.5$	$Entropy_p = -\frac{20}{40}\log_2(\frac{20}{40}) - \frac{20}{40}\log_2(\frac{20}{40}) = 1$
$GiniT_L = 1 - ((\frac{15}{20})^2 + (\frac{5}{20})^2) = 0.375$	$EntropyT_L = -\frac{15}{20}\log_2(\frac{15}{20}) - \frac{5}{20}\log_2(\frac{5}{20}) = 0.811$
$GiniT_R = 1 - ((\frac{5}{20})^2 + (\frac{15}{20})^2) = 0.375$	$EntropyT_R = -\frac{5}{20}\log_2(\frac{5}{20}) - \frac{15}{20}\log_2(\frac{15}{20}) = 0.811$
$I_G = 0.5 - \frac{20\times0.375+20\times0.375}{40} = 0.125$	$I_G = 1 - \frac{20\times0.811+20\times0.811}{40} = 0.189$

Table 4. Information Gain for Tree A

Gini	Entropy
$Gini_p = 1 - ((\frac{20}{40})^2 + (\frac{20}{40})^2) = 0.5$	$Entropy_p = -\frac{20}{40}\log_2(\frac{20}{40}) - \frac{20}{40}\log_2(\frac{20}{40}) = 1$
$GiniT_L = 1 - ((\frac{10}{10})^2 + 0) = 0$	$EntropyT_L = -\frac{10}{10}\log_2(\frac{10}{10}) - 0 = 0$
$GiniT_R = 1 - ((\frac{10}{30})^2 + (\frac{20}{30})^2) = 0.444$	$EntropyT_R = -\frac{10}{30}\log_2(\frac{10}{30}) - \frac{20}{30}\log_2(\frac{20}{30}) = 0.918$
$I_G = 0.5 - \frac{20\times0+30\times0.44}{40} = 0.167$	$I_G = 1 - \frac{10\times0+30\times0.918}{40} = 0.315$

Table 5. Information Gain for Tree B

The Scikit-Learn implementation, however, differs little from the original Random Forest as proposed by Breiman. Scikit-Learn implementation of Random Forest combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class [26]. It is done to improve accuracy and control over-fitting [68].

The Random Forest implementation in our study is performed by making use of the Scikit-Learn library. The Random Forest is implemented using the K-Fold cross-validation technique to prevent over-fitting along with hyperparameter optimization using Grid Search. To perform this task we make use of the Scikit-Learn GridSearchCV method that does both Grid Search and K-Fold cross-validation. Figure 9 shows the architecture of the Random Forest algorithm used in our study. The preprocessed input data is first split into training and testing data. The split training data is used for training the Random Forest algorithm and the testing data is used for evaluating the trained model or algorithm. The input training data is further split into train and test set and passed to the Random Forest algorithm for each fold of the K-Fold cross-validation. Further, the hyperparameter optimization is performed by the Grid Search technique using hyperparameters shown in Table 2. The final result obtained from it is then evaluated for the result. The evaluation is performed using the testing data. Based on the evaluation with the testing data, we record the Precision, Recall Accuracy, and F1 Score as a final result of our algorithm. The detailed implementation of Random Forest architecture is discussed in Random Forest experiment Section 4.2.

[26]https://scikit-learn.org/stable/modules/ensemble.htmlforest

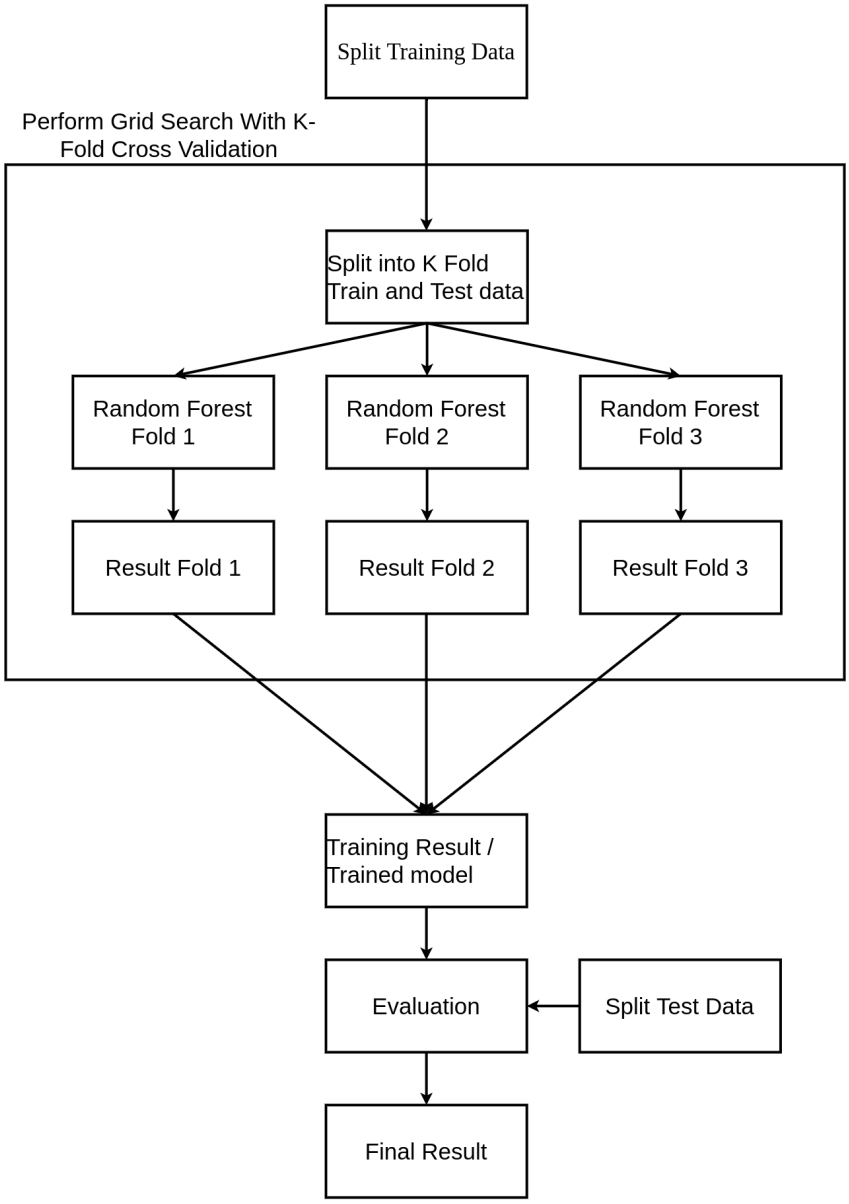

Figure 9. Random Forest Architecture

3.6 Gradient Boosting

The Gradient Boosting machine, another tree-based ensemble method, is the next technique that we adopted for our study. Like Random Forest, it is also one of the robust and competition-winning algorithm [76], [77]. It is used in sentiment analysis [78], predicting delay in flight [79], in medical areas like Predicting RNA-Protein Interactions [80], automatic electroencephalogram (EEG) state recognition that helps in neurological detection [81].

Gradient boosting is an algorithm from the boosting family. The boosting family of algorithms consists of Adaptive Boosting (AdaBoost), Gradient Boosting (GBM), and XGBoost (eXtreme Gradient Boost). The main principle of the boosting based algorithm is to combine the weak learner such that in the end, it forms a strong learner. The combination of the weak learner is performed using the majority vote of every prediction of the weak learner, weighted averaging [78]. For example, in the case of the AdaBoost, it uses a simple averaging technique.

Gradient boosting, one of the members of the boosting algorithm works on the boosting principle by shifting focus towards problematic observations that were difficult to predict in previous iterations and performing an ensemble of weak learners, typically decision trees [76], [77]. It can be applied to both classification and regression problems. Due to the nature of our study, we focus our discussion of Gradient boosting for classification. The model is built iteratively. Each new model is dependent on the previous model. One of the important aspects of the gradient boosting algorithm is that it consists of the differentiable loss function. Figure 10 shows an example of gradient boosting algorithm. Gradient boosting involves three main components [76], [77].

- **Loss Function:** It is used to specify how the loss is to be optimized. It varies based on the problem; classification and regression. Even for classification, multiple loss functions are supported but the default deviance which is negative log-likelihood loss[27], [28].
- **Weak Learner:** Decision trees are used as a weak learner in gradient boosting to make predictions.
- **Additive model:** Trees are added sequentially on each iteration. It is based on the weak learner and it is used to minimize the loss using the gradient descent algorithm.

[27]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[28]https://scikit-learn.org/stable/modules/ensemble.htmlgradient-boosting

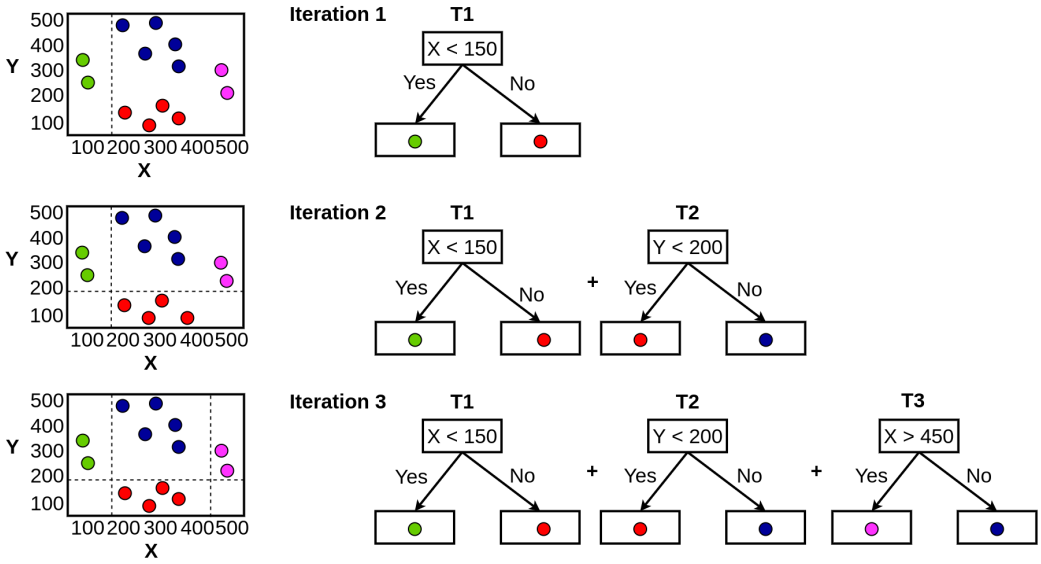

Figure 10. Example visualization of Gradient boosting

The detailed steps for the Gradient boosting classification for the data $\{x_i, y_i\} i = 1...n$ can be stated as follows [76], [77].

- First, we initialize the model with Equation 6 to calculate the log odds. Based on the log odds we calculate the probability of prediction.

$$f_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma) \tag{6}$$

- Then for each m:1 to M, we do the following. This is where the tree construction takes place.
  - We then calculate the Pseudo Residuals using the equation 7, where we calculate the derivative of the loss function with respect to the log odds of the prediction for i=1,2,...N.

$$r_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_m-1} \tag{7}$$

  - After that, we fit a regression tree to the targets r_{im} and label with the region R_{jm}, for each j=1,2,..,J_m and compute γ using Equation 8. Here, we calculate

the output value for the new tree represented by γ. This value is then used to calculate the new log odds and the prediction.

$$\gamma_{jm} = argmin_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma) \tag{8}$$

– We then calculate the new prediction and make update, which is based on the previous prediction using Equation 9.

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{j_m} \gamma_{jm} I(x \in R_{jm}) \tag{9}$$

• Finally, we output the result, $\hat{f}(x) = f_M(x)$, once the stopping criteria are reached.

The architecture of the implemented Gradient Boosting algorithm in our study is shown in Figure 11. Similar to the random forest, the gradient boosting algorithm in our study is also implemented using the Scikit-Learn library using the K-Fold cross-validation technique for preventing over-fitting and hyperparameter optimization using Grid Search. In the case of Gradient boosting, like Random Forest, the prepossessed input data is split into training and testing data. The training data is then passed to the Gradient boosting algorithm along with the hyperparameter shown in Table 3. The training data, however, is further split into train and test data for each fold of the K-Fold cross-validation. On each fold, the algorithm is trained with the split train data of that particular fold and evaluated using split test data of the same fold for accuracy. The final result obtained after grid search and K-Fold cross-validation using Scikit-Learn GridSearchCV is then evaluated with test data. Based on the evaluation with the testing data, we record the Precision, Recall Accuracy, and F1 Score as a final result of our algorithm. The detailed implementation of Gradient Boosting architecture is discussed in Gradient Boosting experiment Section 4.3.

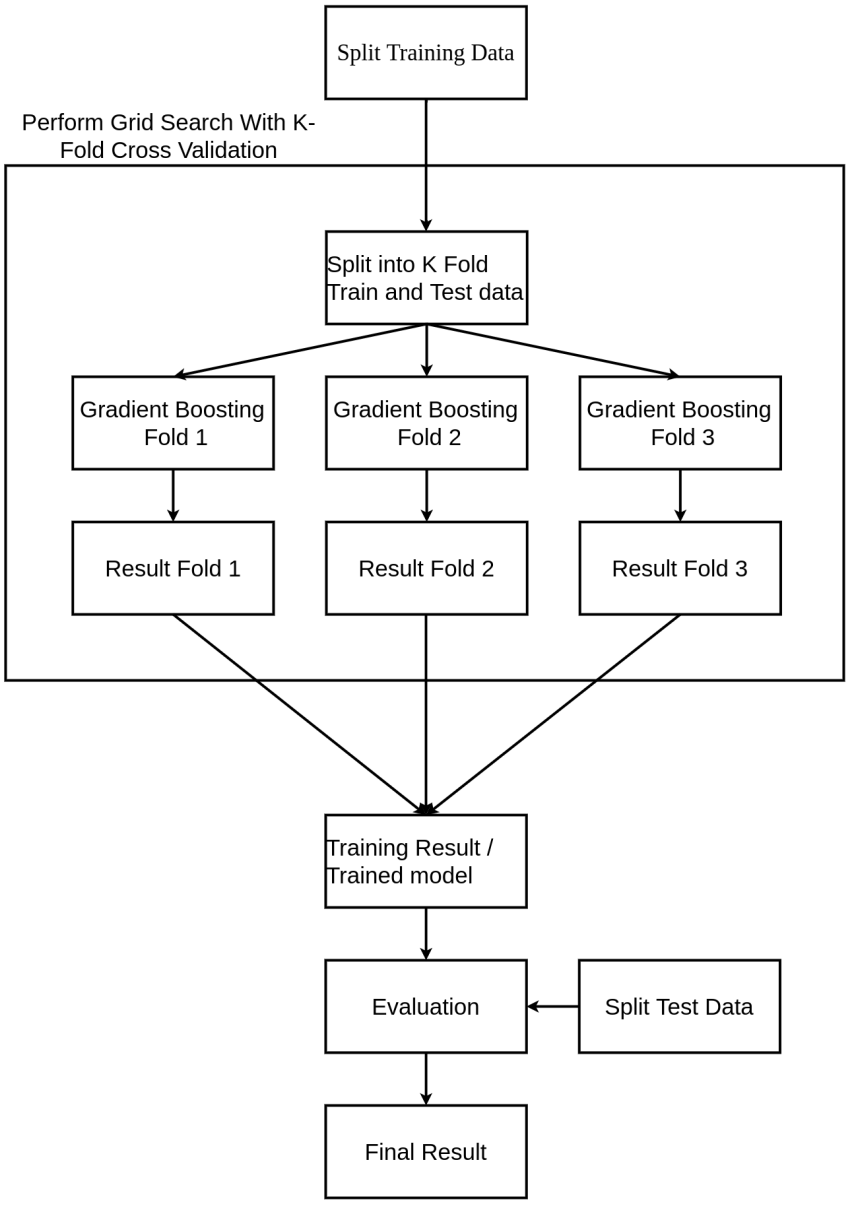

Figure 11. Gradient Boosting Architecture

One of the important task in Gradient boosting algorithm is the construction of the tree and calculation of the γ based on which new log odds and the prediction probability is calculated. Consider an example dataset, shown in Table 6 where we apply the above

algorithm. For the first iteration, we obtain the following tree and γ Figure 12 which can be used for calculation of log-odd using F_0(x) and further the prediction probability. Similarly, for the second iteration, we have Figure 13. Based on the earlier tree and value of F_n(x), we calculate the new log-odds for second iteration F_2(x) as shown in Figure 14. This structure of the tree increases with the increase in a number of iteration. The detail steps of calculation are shown in the Appendix Section 6.

Server	**smart_198**	**smart_3**	**Fail?**
A	19	9	Yes
B	100	98	No
B	61	9	Yes

Table 6. Example Data-set for Gradient Boosting

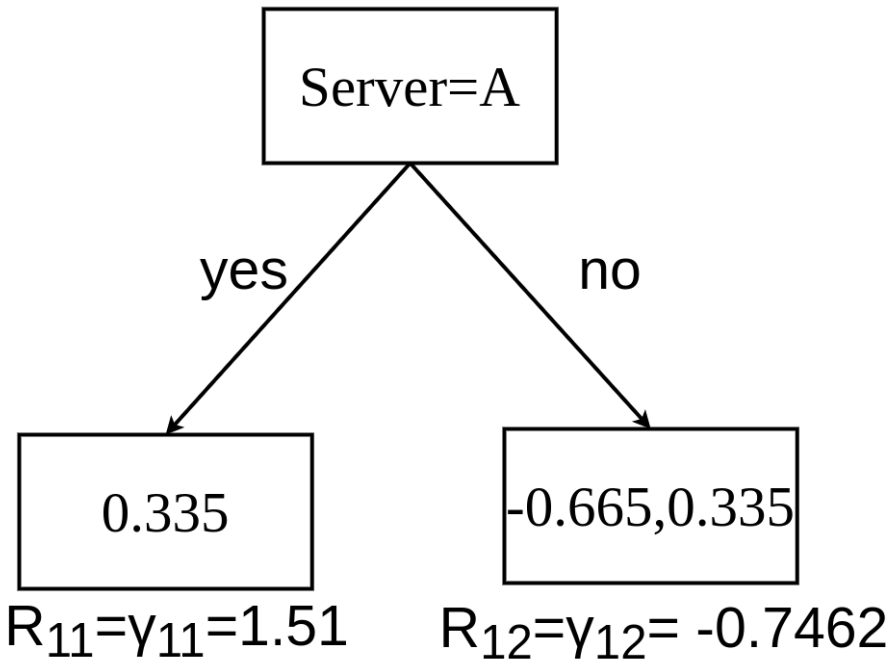

Figure 12. Gradient Boosting Tree for iteration 1

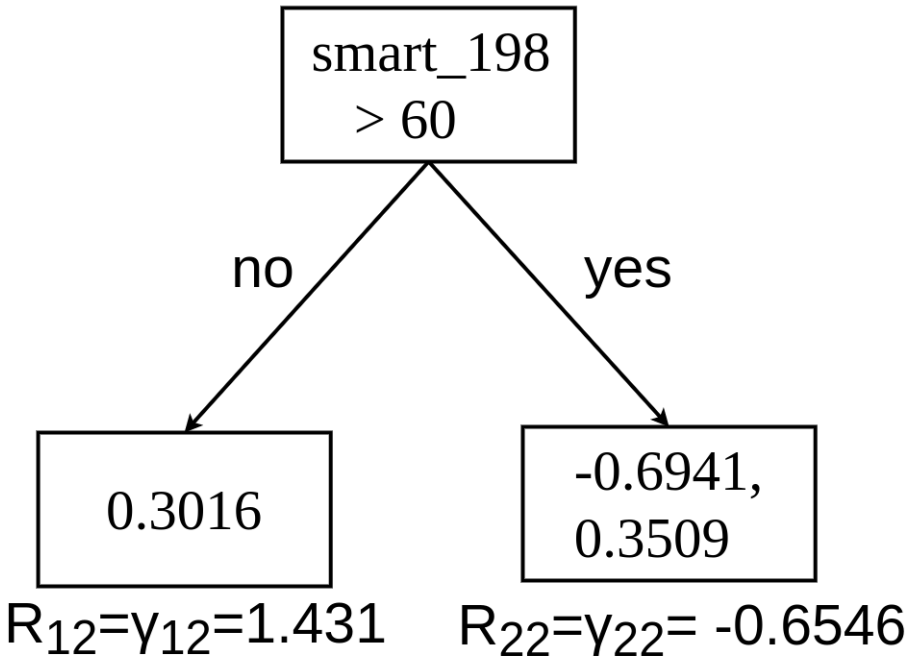

Figure 13. Gradient Boosting Tree for iteration 2

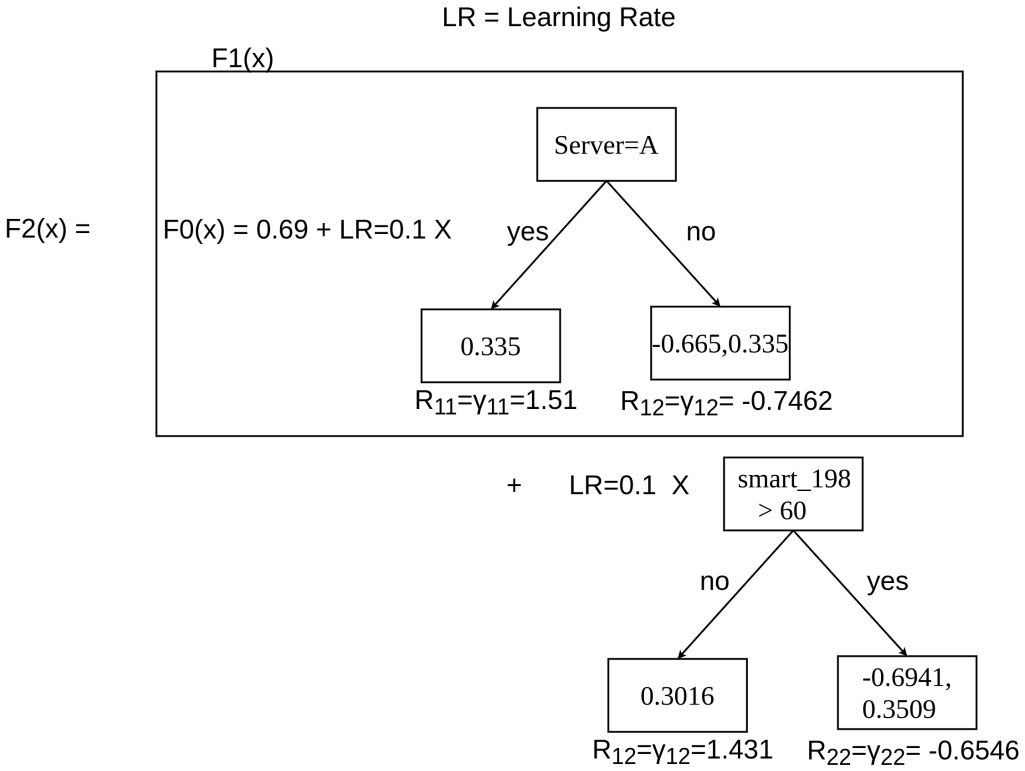

Figure 14. Gradient Boosting F_2(x)

3.7 Recurrent Neural Network

Deep learning over the years has been shown to outperform previous state-of-the-art machine learning techniques [82]. Due to its performance, it is today used to solve some of the complex tasks in different areas like computer vision, autonomous systems, climate analysis [83]. In our study, we also make use of recurrent neural networks-based deep learning techniques.

Neural networks make assumptions of data independence and it breaks in case of the sequential data [84]. This is because in the sequential data there is a dependency on the current state with the previous state. For example, consider the following sentence "the dog is black". Here the order of appearance that is a sequence of the word is important. Such kinds of problems involve text, speech, time series where there is a dependency of state t on the t-1 state. Using the normal neural network, which acts like a mapping function, where a single input is associated with a single output [85], we cannot guarantee it for the correct prediction. Further, if we take the case of the time-series

data for weather forecasting. The information from the previous weeks or months or even years is important for prediction. With the standard neural network also referred as feed-forward neural network, it is impossible to take such temporal conditions into account [86]. A recurrent neural network is (RNN) the special neural network that is designed to overcome such a problem of standard neural networks.

RNN incorporates new design architecture to solve the problem of neural networks. It incorporates the dependence by having a hidden state, or memory, that holds the essence of what has been seen so far [84]. Figure 15 shows the simple RNN architecture and its consecutive unfolded states over the sequence. The U, V, and W are the weight metrics initialized randomly or with a distribution such as normal or Gaussian distribution [85] and are shared among all the states. U is the input weight, W the recurrence weight, and V the output weight. It is because in RNN the same function is applied over and over in a recurrent manner. The states of the RNN can be mathematically represented with Equations 10, 11 [87].

$$h_t = tanh(W \times h_{t-1} + U \times x_t + b_h) \tag{10}$$

$$y_t = softmax(V \times h_t + b_v) \tag{11}$$

The h_t in the above equations represents hidden vector h at time t, x_t input at time t, nonlinear function tanh, and the output vector y_t obtained by application softmax activation function. The tanh nonlinearity helps tackle the issue of vanishing gradient. It is because tanh has very slow decay of the second derivative to zero and that helps keep the gradients in the linear region of the activation function [84]. The tanh determines the final value of the current memory h_t and the output y_t and maps the result value from several equations in a range of 0-1 [85]. The b_h and b_v represents the bias.

The RNN also offers two variations; stacked and bidirectional RNN. The stacked RNN helps to build the multilayer RNN. This stacked RNN solves the issue of not being able to learn properly due to not having enough layers. The other is bidirectional which offers the two-loop working in both directions and is useful for language modeling where both preceding and succeeding words have meaning [88]. Figure 16 shows stacked and bidirectional RNN respectively. Further RNN supports different combinations; one-to-one, many-to-one, many-to-many (sequence-to-sequence) models [88].

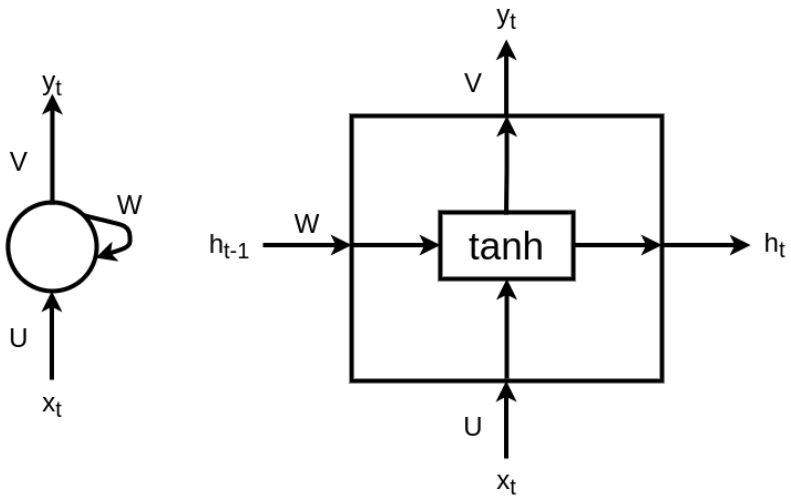

Figure 15. Simple Recurrent Neural Network

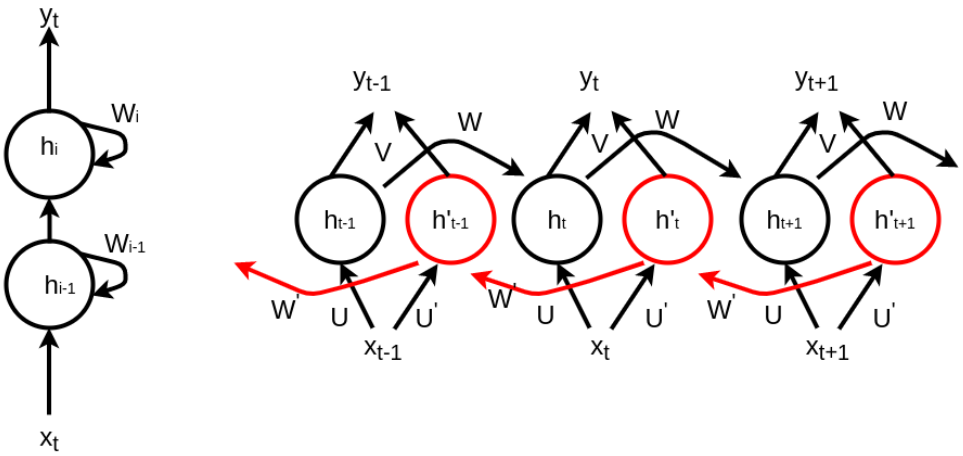

Figure 16. Stacked RNN (Left) Bidirectional RNN (Right)

The RNN is trained using Backpropagation through time algorithm where the network is unfolded for a certain number of steps over time [88]. The RNN however, suffers from the problem of vanishing and exploding gradient. The vanishing and exploding gradient issue occurs when $|W| < 1$ and $|W| > 1$ or eigenvalue $\rho < 1$ and $\rho > 1$ respectively for the scalar and matrix representation of the weight W [88]. When the accumulation of the gradient grows large such that it goes out of range, we call this problem as the exploding

gradient. This makes the weights to be NaN so that it can no longer be updated. And when the gradient decays over time and becomes very small it is overshadowed by the recent gradient making it unable to look back and remember the history efficiently. This is one of the common issues and difficult to detect as the training will still work and the network will produce valid outputs [88].

3.8 LSTM

RNN suffers from the problem of exploding and vanishing gradient for long-term dependency. RNN is only focused on learning but not selectively forgetting [86]. It is required to remember only important things and forget things that are not important. Further, it also helps memory optimization. Long-short-term-memory (LSTM) to solve this problem of RNN introduced new architecture with additional computational units. It is also a drop-in replacement of Simple RNN, meaning the simple RNN can be replaced with LSTM without any side effects, generally yielding better results [84]. The additional computational units involve the gates and explicit state named cell state represented by C_t. The cell state C_t is a memory that is used to hold the information. The introduced gates are input gate, output gate, and forget gate. Figure 17 shows the LSTM architecture. Similarly, the computational steps involved in the LSTM is given by the Equations 12 - 17 [87].

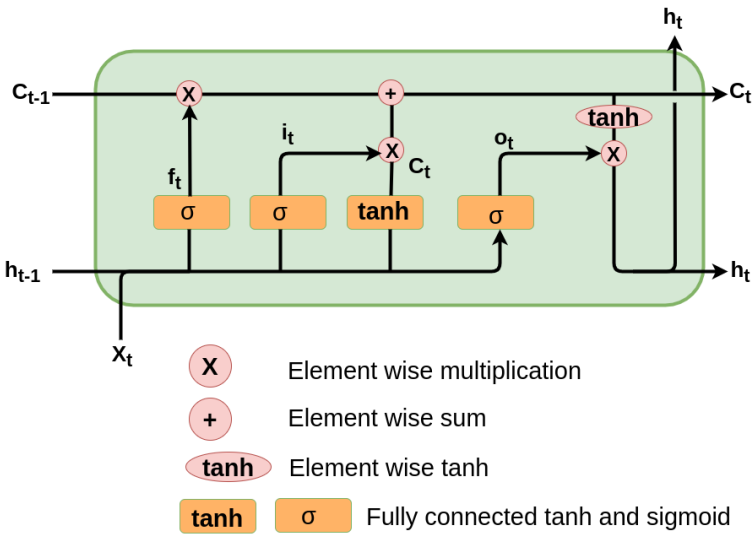

Figure 17. LSTM cell

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{12}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{13}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{14}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{15}$$

$$h_t = o_t \odot tanh(C_t) \tag{16}$$

$$Output(y_t) = softmax(Uh_t) \tag{17}$$

where in the Equations 12 - 17,

- i is the Input gate
- f is Forget gate
- o is Output gate
- C is Cell state
- x represents input
- h represents the hidden state
- W represnts the weights
- b is the bias
- $x*$ for weight W is the input-to-hidden layer and $*$ represents f,i,o
- $h*$ for weight W is the hidden-to-hidden layer and $*$ represents f,i,o
- $\odot$ is element wise multiplication
- t is the time
- U is the hidden-to-output layer weights

The cell state C_t is also referred to as memory state as it stores the information. The information in C_t can be explicitly written or removed so that it stays constant if no outside interference [88]. It is the most important state. It is responsible for the handling the dependencies by remembering and for the actual output [88]. What information to keep and remove is controlled by the gates based on the value of the sigmoid function. The sigmoid function returns either 0 or 1. The 0 value means information is no longer required or can be removed. The same case with the opposite relation holds for the value 1. The sigmoid always returns positive values 0 to 1 but we also need to allow negative

values. For this, we make use of hyperbolic tangent $tanh$ activation in C_t which returns the value within range -1 to 1 allowing symmetry and better performance [86], [87].

The LSTM has three gates namely forgot, input, and output gate. These gates in the LSTM controls the amount of information flow through the current layer to the next layer and also what to remember and forget. The forget gate as the name suggests forgets the information. It is used to remove the information from the cell state. The input x_t multiplied with weight W_{xf} and previous hidden state h_{t-1} with weight W_{hf} are summed together and sent to the sigmoid function. The sigmoid then returns either 0 or 1 value based on which the forget gate takes the action whether to remove or not. The next is the input gate where we pass everything the same sigmoid function to the forgot gate with the parameter changed. It is also modulated by the sigmoid value. Based on the value of the sigmoid, it updates or stores the new value to the cell state. The output gate, which again is controlled by the sigmoid function decides what value to take from the cell state to give it as an output. This value is then used to get the hidden state along with C_t as shown in Equation 16. The hidden state value is then used with the softmax function to get the result using Equation 17.

The implemented LSTM architecture in our study is shown in Figure 18. Our LSTM implementation consists of the 8 LSTM layers and 6 Dense layers followed by one input and one output layer. The implemented LSTM layer consists of the 2048 hidden units and the Dense layer consist of 1024 units. Further, each LSTM layer incorporates the activation function ReLU, dropout layer, and L2 regularization for the kernel regularizer. Like other algorithms, LSTM is trained with split training data and tested with the test data. The training data from the input layer passes through each layer until it reaches the final output layer. The output layer is a sigmoid activated dense layer with 1 hidden unit. Similar to Random Forest and Gradient boosting, LSTM is also evaluated against Accuracy, Precision, Recall, and F1 Score. Further, the detailed implementation of LSTM architecture is discussed in LSTM experiment Section 4.4.

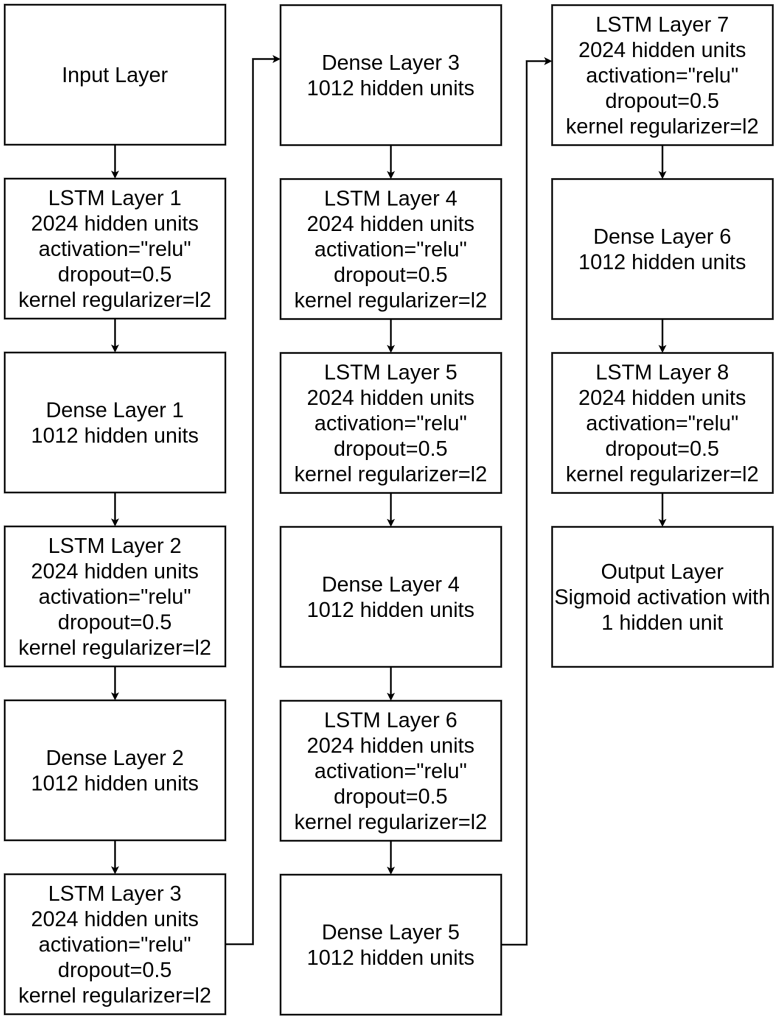

Figure 18. Implemented LSTM Architecture

3.9 GRU

Gated Recurrent Unit (GRU) is another RNN based model that is designed to combat the long-term dependency issue like LSTM. It is a variant of LSTM introduced in 2014 by Cho et al. [89]. LSTM requires updating a lot of parameters in every iteration during the backpropagation phase, increasing the training time [87]. This is primarily because of the presence of many gates and states. GRU was designed keeping this in mind to simplify this complexity of LSTM. So we could call GRU a simplified version of LSTM.

The simplification is achieved by reducing the number of gates and states. GRU cell has only two gates namely reset and update gate and one hidden state. Figure 19 shows the architecture of GRU. Similarly, the computational steps involved in GRU forward propagation is given by the Equations 18 - 21 [87]. In the backward propagation, the derivative of loss is calculated with respect to all weights using gradient descent and updated [87]. It is done to minimize the loss.

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{18}$$

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{19}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{20}$$

$$Output(y_t) = softmax(Uh_t) \tag{21}$$

where in the Equations 18 - 21,

- x represents input
- h represents the hidden state
- o represents output
- W represnts the weights
- b is the bias
- $x*$ for weight W is the input-to-hidden layer and $*$ represents z,r
- $h*$ for weight W is the hidden-to-hidden layer and $*$ represents z,r
- $\odot$ is element wise multiplication
- z is update gate
- r is reset gate
- U is the hidden-to-output layer weights

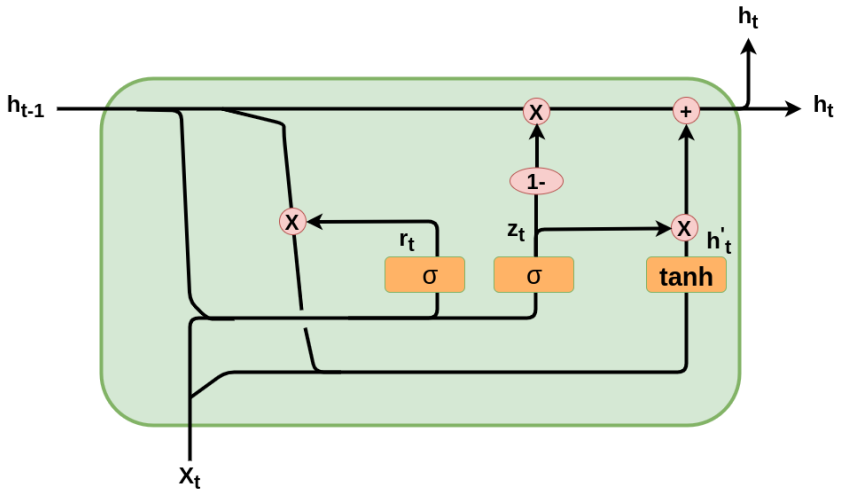

Figure 19. GRU cell

The reset gate r which is sigmoid function of sum of the product of W_{xr}, x_t and W_{hr}, h_{t-1} is similar to LSTM forget gate. It decides the degree to which the content of the previous output to be preserved based on the value of the sigmoid function. The sigmoid like in LSTM controls the gate in GRU. However, the reset gate is not enough to determine the right output with enough accuracy, considering both short and long-term dependencies so an update gate has been added to increase expressivity [86]. The update gate z is also the sigmoid of sum of the product of W_{xz}, x_t and W_{hz}, h_{t-1}. The update step decides based on the sigmoid function value what information to take forward to the next step. The complement of update gate z_t while updating the hidden state helps to avoid the new gate as shown in Equation 20. After the calculation of the h_t, the final result is obtained by applying a softmax activation function as in Equation 21.

GRU, a simplified version of LSTM is also the drop-in replacement for the simple RNN [84]. GRU does not have persistent cells for storage like in the case of LSTM. In terms of the performance, studies have shown both GRU and LSTM to have a comparable performance [84], [86]. GRU due to its simplified structure speeds up the training process. However, given enough data an LSTM's greater expressive power may lead to better results [84].

The implemented architecture of GRU in our study is shown in Figure 20. The implementation of GRU like LSTM consists of 8 GRU layers and 6 Dense layers. Each of the GRU layers incorporates the activation function ReLU, dropout layer, and L2 regularization for the kernel regularizer. Similar to other algorithms we train our GRU model with training data that will be passed through each layer until it reaches the final

output layer. The output layer is also a dense layer with 1 hidden unit and activated by the activation function Sigmoid. The use of different activation functions is described in Section 3.10. The activation function, dropout layer, and the regularization are used to prevent over-fitting. The output of each previous layer acts as the input to the next layer. Similar to Random Forest, Gradient boosting, and LSTM, GRU is also evaluated against Accuracy, Precision, Recall, and F1 Score. Further, the detailed implementation of GRU architecture is discussed in GRU experiment Section 4.5.

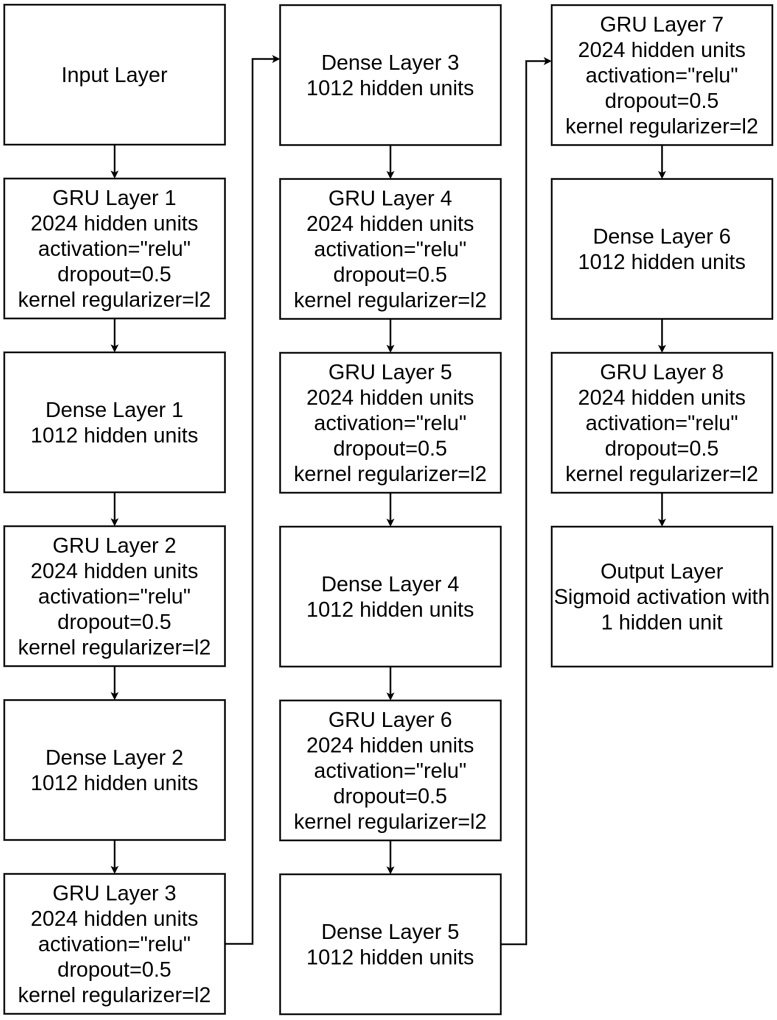

Figure 20. Implemented GRU Architecture

3.10 Activation Function

Deep learning algorithms are multi-level representation learning techniques that allow simple non-linear modules to transform representations from the raw input into the higher levels of abstract representations, with many of these transformations producing learned complex functions [90]. It is inspired by the human brain and works similarly by activating the neurons. Based on which neuron is activated the necessary steps are taken in the algorithm. How the neurons are activated is controlled by a function called an activation function. An activation function, also known as a transfer function is also used to introduce non-linearity [87]. It is necessary to apply an activation function to learn the complex patterns from data. The activation function adds the non-linearity, which without activation function resembles linear regression [87]. There are over 20 activation functions [90] but we shall only discuss the ones relevant to our study.

3.10.1 Sigmoid

Sigmoid is one of the most commonly used activation function. It is also referred to as a logistic function or squashing function [90]. It gives the probability value within the rage of 0-1. It is used in the output layer for making a prediction in the case of binary classification. Further, the sigmoid is differentiable, meaning that we can find the slope of the curve at any two points and monotonic, which implies it is either entirely non-increasing or non-decreasing [87]. Equation 22 is used in Sigmoid activation function [87]. The Sigmoid, however, suffers major drawbacks which include sharp damp gradients during back-propagation from deeper hidden layers to the input layers, gradient saturation, slow convergence and non-zero centred output thereby causing the gradient updates to propagate in different directions [90]. The other variants of Sigmoid include Hard Sigmoid, Sigmoid-Weighted Linear Units, Sigmoid-Weighted Linear Units, Derivative of Sigmoid-Weighted Linear Units, and Nwankpa et al. provide detailed description [90].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{22}$$

3.10.2 Softmax

The softmax activation function is the generalization of the Sigmoid [87]. It is represented by Equation 23 and also gives the probability [87]. It is like Sigmoid used in the final layer for making a prediction in case of the multi-class classification.

$$f(x_i) = \frac{e^i}{\sum_j e^{x_j}} \tag{23}$$

3.10.3 Hyperbolic Tangent Function (Tanh)

The hyperbolic tangent function is a smoother non-linear activation function [91] that is centered around 0 with the value ranging from -1 to 1. Tanh also resembles the S-shaped curve like sigmoid and is differentiable and monotonic [87]. The tanh like sigmoid does not solve the problem of vanishing gradient and also produce dead neuron as it can only attain gradient of 1 when input is 0 [90]. The main advantage provided by the function which is represented by Equation 24 is that it produces zero centred output thereby aiding the back-propagation process [90].

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{24}$$

3.10.4 ReLu

ReLU eliminates the problem vanishing gradient problem of sigmoid and Tanh and is used in the hidden layer of the networks [90]. It is the most popular non-linear function is the rectified linear unit (ReLU), which is simply the half-wave rectifier [91] represented by Equation 25 [87]. ReLU typically learns much faster in networks with many layers, allowing training of a deep supervised network without unsupervised pre-training [91]. It outputs the value from 0 to infinity range. From the Equation 25 it is clear that anything negative will be turned to zero, meaning the neuron will be dead. This snag for being zero for all negative values is a problem called dying ReLU [87]. The other variants of ReLU are Leaky ReLU and Exponential linear unit (ELU), which solves this dying ReLU problem by introducing a small negative slope [87].

$$f(x) = \begin{cases} 0 & \text{x} < 0 \\ x & \text{x} \geqslant 0 \end{cases} = max(0, x) \tag{25}$$

3.11 Loss Function

In this section, we will discuss the essence of the loss function. Like other discussions, in this section also we will only focus on the one related to our study. The loss function discussed in this section is used in our deep learning model.

Loss functions are another essential building block of neural networks, and they measure the difference between our predictions and reality [92]. Therefore, we always try to minimize the loss function to the best possible lowest value. In other words, we would like to find minimum points in loss functions called global minima especially while creating neural networks [92]. Different types of loss functions are available and they are used depending on the nature of the problem. For example mean square error (MSE) used for a regression problem, cross-entropy for classification problems.

The cross-entropy loss function for the true label y and predicted label $\hat{y}$ is defined as in Equation 26 [93]. Further for binary classification problems, it is also called Bernoulli's negative log-likelihood and Binary cross-entropy [92]. It is represented by Equation 27. And in the case of the multiclass classification, it can be generalized as shown in Equation 28.

$$loss(\hat{y}, y) = -\sum_{i=1}^{n} y_i log(\hat{y_i}) \tag{26}$$

$$loss(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\hat{y} + (1 - y_i) log(1 - \hat{y_i}) \tag{27}$$

where,

- n is the number of iterations.
- y_i is the actual value and for binary classification it is 0,1
- $\hat{y_i}$ is the predicted probability P(y=1 or 0 | x = x_i)

$$loss(\hat{y}, y) = -\sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij} log(\hat{y_{ij}}) \tag{28}$$

3.12 Summary

In this chapter, we introduced the approach taken for implementation. We first introduced the architecture. Thereafter, the data collection describing steps and tools used for data collections and what data was collected. Moving on we also saw how data was preprocessed and the need for it. After that, we presented the selected algorithms for our study. We then discussed why such selection was made and how the algorithm works in detail. Moreover, we also showed how an algorithm can be optimized using a hyperparameter optimization technique. We then end this chapter with a discussion of activation function and loss function. In the next chapter, we shall see how the experiment was performed.

4 Experiment

This section describes the detailed implementation of the model and the experimentation. The experimental setup is first described in Section 4.1, including all the details of the tools as well as the system configuration. Further, we discuss how the experiment was performed with Random Forest in Section 4.2, Gradient Boosting Classifier in Section 4.3, LSTM in Section 4.4 and GRU in Section 4.5, with all the details including the hyperparameters used, training performed.

4.1 Experimental Setup

One of the most time-consuming processes in a machine learning-based algorithm is training. The time of the training depends on algorithm implementation, data size, training strategy, and computing power. The training process benefits with higher computational power. In short, it can be said that the higher the computation power faster the training. This is the reason for performing training by using the services from the High-Performance Computing centers, Cloud providers like AWS, Google Cloud platform utilizing their high computation power. Realizing this, we selected the system with high computational power that we could make use of. The System used in our study consist of 62 GB of Random Access Memory (RAM) with 16 core Intel i7 3.8 GHz processor. Further, the system is equipped with two Nvidia RTX 2080 Ti Graphics Processing Unit (GPU). The GPU is having 4352 CUDA cores with 11 GB GDDR6 Standard Memory and it supports the Base Clock 1350 MHz for CUDA cores and 14 Gbps memory speed and 616 GB/sec memory bandwidth [29].

The machine learning-based algorithms can be implemented in multiple languages. In our study, we selected Python as the choice of our programming language. The selection of Python as the programming language is based on the simplicity as well as the popularity of the language in the area of data science. The other reason is the availability of the python-support of popular machine learning and deep learning frameworks and libraries like Tensorflow, Keras, Scikit-Learn.

Scikit-Learn a python based open-source machine learning library is selected for the implementation of Random Forest and Gradient Boosting Classifier. Scikit-Learn version 0.22 is used in our study. Similarly, for the implementation of the deep learning algorithm LSTM and RNN, we selected an open-source deep learning framework TensorFlow version 2. Tensorflow which makes use of the Keras as high-level neural network API for building and training deep learning models[30] uses the available GPU with no code changes[31]. Nvidia provides a C++ based library TensorRT to facilitates high-performance

[29] https://www.nvidia.com/en-eu/geforce/graphics-cards/rtx-2080-ti/

[30] https://www.tensorflow.org/guide/keras

[31] https://www.tensorflow.org/guide/gpus

inference on NVIDIA graphics processing units (GPUs)[32]. It is used to optimize and accelerate the dep learning training process. Moreover, TensorFlow has it integrated [32]. So, in order to make full utilization of the available GPU and to optimize and accelerate the deep learning training process, we also use TensorRT. The TensorRT version 6.0.1 was used in our study. Furthermore, TensorRT requires CUDA, a parallel programming platform. So to facilitate the TensorRT, we installed CUDA Version 10.0.130.

Further, the preprocessed input data in our study is split into the train and test. During the experiment, we tried different combinations of train and test data like 55%:45%, 70%:30%, and 80%:20% before 60%:40%. The experimentation with the different combinations is because of not getting satisfying results. However, we only recorded the result from the 60%:40% split of training and testing data as it was giving the best result for us as discussed in Section 5. Moreover, studies [94] also have shown 60%:40% split to be better. This was based on the similar kinds of observations that were made in studies like [33], [45], [39] where the reported result is from only one combination of train and test split. Following similar patterns, we recorded the best result only in our study. The training data in a later section of our discussion shall mean the split 60% of training data and test data shall refer to the 40% test data.

4.2 Random Forest

This section describes how the implementation of the Random Forest algorithm was made in our study. Random Forest can be used for both regression and classification. In our study, we use the classification version of it. The implementation of the algorithm is based on the Scikit-Learn library which provides an implementation of different machine learning libraries. The detailed implementation architecture of the Random Forest used in our study is shown in Figure 9. We first begin the experiment by providing the input data. The input data here is the pre-processed training data that has been separated to train the algorithm.

The next step before beginning the training is to set the hyperparameters. Random Forest and other machine learning algorithms come with input parameters called hyperparameters. The hyperparameters also come with a fixed set of values. This means in an experiment if we do not set any hyperparameters, the default one will be used. Information regarding the hyperparameters of Random Forest used in our experiment can be found from the official documentation [33]. The hyperparameters are altered according to the need of the experiment.

The selected hyperparameters for our study are shown in Table 2. Similarly, Table 7 shows the default value for our selected hyperparameters. To tune the algorithm for

[32] https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html

[33] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

maximum accuracy, we set selected hyperparameters with the list of values as in Table 2. The selection of the hyperparameters is made based on the experience. Scikit-Learn GridSearchCV has inbuilt support to it and can be passed as an input parameter.

Parameter Name	**Value**
n_estimators	100
max_features	'auto'
criterion	'gini'
min_samples_split	2

Table 7. Random Forest Default Hyperparameter value

Once we have the set of hyperparameters to tune, the next step is to select the strategy for tuning the hyperparameters. This means among available techniques like Grid Search, Random Search, Bayesian optimization [57] which one to use to find a suitable set of hyperparameters for our experiment. In our experiment, we selected Grid Search as it is simple and does an exhaustive search.

Over-fitting is one of the major problems in machine learning. And if it is not dealt with properly then we cannot achieve the desired result. So we use the K-Fold cross-validation technique. It is used to control over-fitting. In the K-Fold cross-validation, K is the number of folds and is defined at the beginning of the experiment (training process). In our study, we set the value of K to 3. The selection of the value of K in our research is based on the trail and experiment. Scikit-Learn offers a method GridSearchCV[34] that is capable of performing hyperparameter tuning using grid search and also the K-Fold cross-validation at the same time. We, in our implementation, make use of the same GridSearchCV method from Scikit-Learn Library.

The training in machine learning is a time-consuming task. Moreover, with the use of grid-search, which performs an exhaustive search with all hyperparameters, it makes the process of training slower. The use of parallelization can, however, improve the process. Further, the system is available for the experiment suitable for doing this kind of task. To take advantage of available hardware and speed up the training process, we use joblib with a value set to 12 in our implementation. Joblib is a set of tools to provide lightweight pipelining in Python providing services like transparent disk-caching of functions and lazy re-evaluation and easy simple parallel computing[35].

[34] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

[35] https://joblib.readthedocs.io/en/latest/

The random state helps to maintain the replicability of the result. In our experiment, we have set the random state to be 42. It was passed as an input parameter to the algorithm Random Forest. With everything set, we then train our algorithm. Once the training process is complete, we test the trained model with the remaining 40% test data which is discussed in Section 5. Table 8 shows the tuned value of hyperparameters with which we obtain the best result for our experiment.

Parameter Name	**Value**
n_estimators	300
max_features	'auto'
criterion	'gini'
min_samples_split	8

Table 8. Tuned Random Forest Hyperparameter value

4.3 Gradient Boosting

The next algorithm that we used in our study is the Gradient Boosting algorithm. In our implementation, we use the GradientBoostingClassifier[27] which is the classification implementation of the Gradient Boosting algorithm. The implementation like Random Forest is based on the Scikit-Learn library. Figure 11 implemented architecture of Gradient Boosting algorithm.

Like Random Forest, we begin first with input training data. But before moving to the process of training we do other necessary setups. We begin by setting the list of the hyperparameters that will be used to tune the algorithm. Table 9 shows the default value of the hyperparameters that we have considered in our study. Similarly, Table 3 shows the list of values that were selected for tuning our gradient boosting algorithm.

The hyperparameter learning rate is very important because it determines how learning should take replace that is the steps in learning. The selection of the list values here makes a difference and can either be achieved with experience, trial, and experiment or from the study of established research. In our case, it was an experience that proved to be helpful. Further, the loss hyperparameter is used for the optimization of loss and is discussed in detail in Gradient Boosting methodology Section 3.6.

Parameter Name	Value
loss	deviance
learning_rate	0.1
max_features	None
min_samples_split	2
n_estimators	100

Table 9. Gradient Boosting Hyperparameter Default Value

The next step after hyperparameter is to select a technique for hyperparameter optimization. Following the notion of Random Forest, we use the Grid Search technique to select the best hyperparameters from the given search grid. Moreover, similar to Random Forest, we also adopted the K-Fold cross-validation technique that helps to control over-fitting with the value of K being 3. The number likewise to Random Forest is selected based on trial and experiment. The implementation of K-Fold cross-validation and grid search was made using GridSearchCV.

We then set the random state for value for the algorithm and pipelining through joblib to make the training process faster and to take benefit of available hardware. The random state was set to 42 and joblib value to 12. With these things set, we train our algorithm with the training data. Once the training is complete, we then evaluate with remaining 40% test data. Table 10 shows the value of the tuned hyperparameter with which we were able to obtain the best result for a Gradient Boosting algorithm in our study.

Parameter Name	Value
loss	deviance
learning_rate	0.01
max_features	5
min_samples_split	3
n_estimators	500

Table 10. Gradient Boosting Hyperparameter Tuned value

4.4 LSTM

In this section, we will discuss the first deep-learning-based algorithm LSTM that we used in our study. Like any other machine learning algorithm, we first begin by reading the input data and thereby proceeding to further steps. Since we are working on time series data and LSTM which is specifically designed to work with time series varies than classical machine learning algorithms like Random Forest, Gradient Boosting Machine, and so on. The input has to be passed in the N×T×D form where N represents the number of samples, T represents time shift window and D is the features. In our study, we use the 10 as the time stamp window. The number of input features D was set dynamically to take it from input data. It was set to a second value that was returned by the shape method of the data-frame. N was set to the total length of the input data minus the T. The data is then pre-processed to standardize and handle the missing value.

The pre-processed input is then passed to the model. The input that is passed to the input layer of the LSTM model is of the shape T, D. Our LSTM implementation consists of the 8 LSTM layers and 6 Dense layers followed by one input and one output layer. Figure 18 shows the implemented architecture of the LSTM. The LSTM consists of the 2048 hidden units and the Dense layer consist of 1024 units. Further, each LSTM layer incorporates the activation function ReLU, dropout of 0.5, and L2 regularization for the kernel regularizer with the regularization value of 0.1. Some of the popular deep learning architectures SigAlexNet, ZFNet, VGGNet, SegNet, GoogleNet, SqueezeNet, ResNet, ResNeXt, MobileNets also uses the activation function ReLU for hidden layer [90] because of its robustness against vanishing gradient problem. The same is the reason for us selecting ReLU as an activation function.

Over-fitting is one of the major problems in both machine learning and deep learning. Even in our experiment phase, we experienced very high over-fitting resulting in us taking appropriate actions. To take appropriate actions to handle we introduced dropout. The dropout acts as a switch turning off the neurons based on the provided dropout value. It is very important because dropout is a proven technique to reduce over-fitting [95]. We also introduced the regularization. The regularization helps to control the model by penalizing it. Further, we also make use of the early stopping to control the over-fitting by using a custom callback. Early Stopping monitors the particular metrics that have been said to monitor and stops the training once the model starts over-fitting. Taking these approaches, we were able to avoid over-fitting in our experiment. Further, we use binary crossentropy as the loss function.

Another important thing is the learning rate. It is the one that decides how much step to take forward in the process of learning. The learning rate is very important because if it is too small then it gets stuck into the local minima. And if it is too higher there will be divergence, both of the situations we would like to avoid. This is because the accuracy of the model depends on how well we learn and the learning rate helps to do so. The fixed learning rate can be set to the model and train with it. This process of setting a

fixed learning rate is not good because it does not take account of the changing situations like increasing or decreasing the loss. We, therefore, would prefer to have something adaptive. In our implementation, we make use of one such adaptive learning rate method Adam with learning rate schedules. We also implemented the learning rate scheduler with the InverseTimeDecay with an initial learning rate 0.001, decay steps 2000, decay rate 1, and staircase False to apply decay in a continuous fashion. The learning rate scheduler callback is called after every epoch and reduces the learning rate to the smaller value. This allows large weight changes at the beginning of the learning process and small changes or fine-tuning toward the end of the learning process [96].

The batch size of 556 and the epoch of 430 was used for training the model. Though TensorFlow makes use of the available GPU, it does not make use of it efficiently unless instructed to do so. By efficiency, we mean the distribution of the load. For example, if you have got multiple GPUs then TensorFlow does not make use of the multiple GPUs in a distributed manner unless instructed otherwise. So to take advantage of the available GPUs in an efficient manner, we use the distributed training strategy. TensorFlow provides different distributed training strategies like MultiWorkerMirroredStrategy, TPUStrategy, CentralStorageStrategy, MirroredStrategy[36]. These strategies are meant to be used based on the placement of GPUs or TPUs. In our case, both the GPUs are present in a single machine and therefore the supporting distributed training strategy MirroredStrategy was used. MirroredStrategy is a synchronous distributed training that creates replica per GPU and communicates using efficient all-reduce algorithms[36].

4.5 GRU

In this section, we discuss the implementation of the second deep-learning algorithm GRU. Figure 20 shows the implemented GRU model in our study. The implementation design of LSTM and GRU is the same as the LSTM unit replaced with GRU. One of the reasons for using GRU along with LSTM as they are both intended for solving the same types of problems is to evaluate their performance as claimed by the studies [84]. Because of this our GRU implementation consists of the same settings as in LSTM implementation. Further, we follow the exact same steps in training and testing of GRU as LSTM.

4.6 Summary

To summarize this chapter, we saw the experimental setup and how the experiment was performed. We presented the programming language, the library used, and also the description of the computational capacity. Further, in the experiment to particular algorithms, we also discussed the selection of hyperparameters, training strategy, measures

[36]https://www.tensorflow.org/guide/distributed_training

to control the over-fitting. In the next chapter, we shall discuss the results.

5 Results

It is by human nature that we always want things better. This quest for better results is one of the reasons for innovation. It is the same reason for the existence of multiple variants of the same technique. For example, let's take one of the popular algorithms used today in autonomous vehicles for tracking and position estimation. The algorithm is the Kalman Filter. There exist different modifications of this algorithm namely Kalman Filter, Unscented Kalman Filter, Extended Kalman Filter. The improvement to the initial version of the algorithm is the reason for multiple versions of the same algorithm. This brings to one of the most important questions, how do we determine what result is better and what is not? Why do we care about it? What result is better and what is not is determined using certain metrics and those metrics are generally called evaluation metrics. The reason we use evaluation metrics to evaluate our algorithm is to understand how good the algorithm is performing and how well it will perform in a similar unseen scenario. Therefore, correct use of model evaluation is vital in academic machine learning research as well as in many industrial settings [97].

There are different evaluation metrics available and its use-case varies depending on the nature of the problem. For example, we use different metrics for regression and classification problems. Our work of failure prediction belongs to the classification category. So we select the classification related evaluation metrics. Accuracy is one of the most commonly used evaluation metrics for the classification task and is one of our evaluation metrics. The other selected evaluation metrics for our work are precision, recall, and F1 score. Accuracy can sometimes be misleading in case of imbalance data [98], [99]. Therefore, for our work, we have selected multiple evaluation metrics.

Accuracy is the measure of overall correct predictions. It can be calculated using the Equation 29 [100]. Precision is defined as True Positive (TP) divided by the sum of TP and FP. It is represented by Equation 30 [100]. It is a measure of a model's exactness and a higher precision value for a classifier is an indication of a good classifier [98]. The recall is True Positives divided by the sum of True Positives and False Negatives, represented by Equation 31 [100]. It is also called sensitivity, or true positive rate. It assesses the effectiveness of the classifier on the positive/minority by measuring the accuracy of positive cases [98]. Further, to understand the balance between precision and recall, we have also taken the F1 Score. F1 Score is the harmonic mean of both precision and recall [101] and is given by the Equation 32.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{29}$$

$$precision = \frac{TP}{TP + FP} \tag{30}$$

$$recall = \frac{TP}{TP + FN} \tag{31}$$

$$F1Score = 2 \times \frac{precision \times recall}{precision + recall} \tag{32}$$

True positive (TP) means that the model predicts positive and the actual value was the same. Similarly, for True Negative (TN), the negative prediction by the model matches the actual value. False-positive (FP) means the model predicts positive but the actual values are negative and the reverse is true in the case of False-negative (FN).

To access the quality we evaluated our work first against accuracy . For more evaluation of our work, we also make a comparison with other similar works. Figure 21 shows the accuracy of our implemented Random Forest, Gradient Boosting, LSTM, and GRU algorithm and its comparison with other studies. From the figure, we can see a very good result from all our implemented algorithms, with Random Forest giving the best. Moreover, the comparison with other similar works [15], [31], [18] and [34] also shows our work to be better.

Accuracy can sometimes be misleading [98], [99] and to not fall in the trap of it we further make an evaluation using precision and recall showed by Figure 22. Some studies like [18] also makes multiple metrics like accuracy, recall for evaluation. Similar to accuracy, we also compare the recall to our study. Results from the precision and recall show that our accuracy is not misleading as we have higher precision and recall with all of our implementations. Ideally, this is what we want, high precision and recall [102].

Similarly, we move further in the evaluation of our work, and finally, we evaluated it against the F1 Score which measures the balance between precision and recall. And like accuracy and recall, we also compare it with other studies [37]. Like the results from other evaluation metrics, it also gives very good results shown in Figure 23. The high value of the F1 Score shows that our result of precision and recall is balanced and further comparison also shows our result to be better.

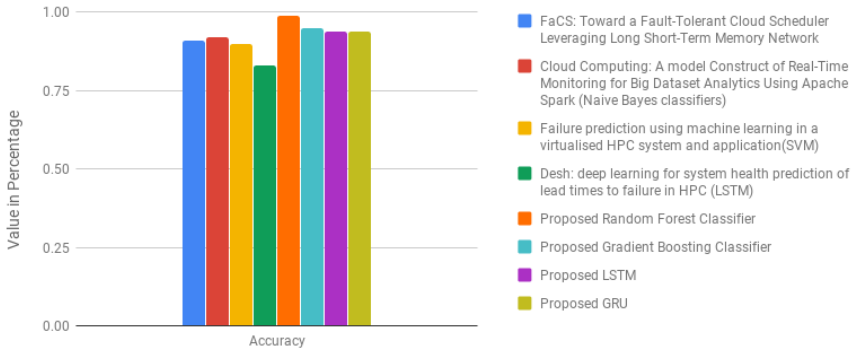

Figure 21. Comparison of Accuracy

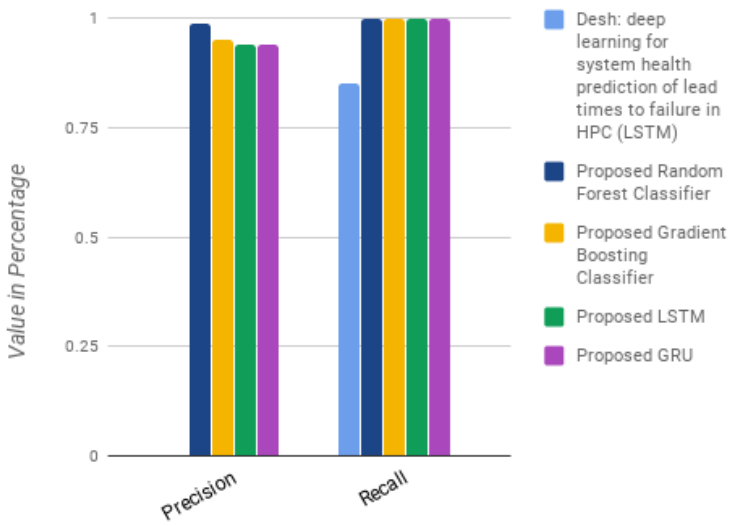

Figure 22. Comparison of Precision and Recall

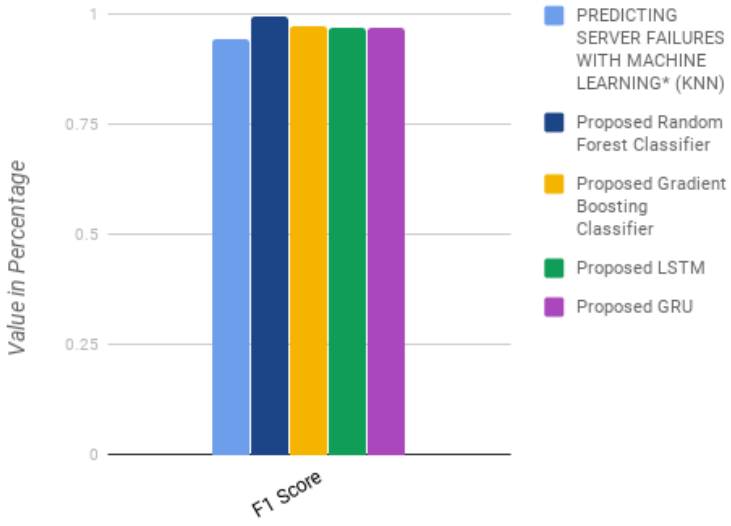

Figure 23. Comparison of F1 Score

5.1 Summary

In this chapter, we presented results from our work and also made a comparison with some relevant state-of-the-art. Moreover, we also discussed the evaluation metric selection and its importance. In the next chapter, we shall see what can be done further to improve this study.

6 Conclusion and Future work

This study investigates the prediction of failure using the combined metrics approach using both classical machine learning and deep learning approach. In this study, we studied and analyzed the metrics that are critical to failure prediction and further made an analysis on it to make early failure prediction. We implemented and tested four different algorithms Random Forest, Gradient Boosting, LSTM, and GRU. Based on our research, the following are the key findings of our study.

- The first is the metrics itself. We made a detailed study of different available metrics and a careful selection from the available metrics as shown in Table 1. The selected metrics have a high impact on the failure which we also have demonstrated by the result.
- We from our study have demonstrated the advantages of using metrics in a combined manner for failure prediction. The use of the combined metrics is tested by implementing different algorithms and the result is validated against similar such studies without using combined metrics.
- Further, we also demonstrated the advantage of using the artificial-intelligence-based technique for making failure prediction unlike rule-based tools like Prometheus.

However, despite the advantages we showed, we find some limitations to our study which in our view could have made the failure prediction more robust. The following are the limitations observed in our study.

- The first limitation was the amount of data available. We were fortunate to have the data with all the metrics available as per our requirement. However, we could only collect the data for a month. If we had more data then our deep learning models could have learned much better. It is because the more data we give to the deep learning algorithms the better it can learn. This is the reason we expect our deep learning algorithms to have less accuracy compared to a classical machine learning approach.
- The second limitation is also related to the data. Unfortunately, the data we have did not consist of the failure record. Therefore, we had to prepare the failure condition following the hardware specification and based on earlier studies. The failure can occur at any time and it necessarily may not occur following the hardware specification. The lack of the failure record we consider to be limited.

Machine learning-based algorithms benefit from more information. With the use of the combined metrics, we provide an algorithm with more information that it can learn from. Because of this algorithm will perform better than having less information. We

have demonstrated by the experiment the advantages of providing more information to the algorithm. However, the lack of data and failure record is the limitation we consider for this study. A future improvement to this study would be to use a similar approach with more data and failure information.

References

[1] D. Sullivan, "The definitive guide to cloud computing," *Real Time Nexus*, no. 1, pp. 4–11, 2010.

[2] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto, *et al.*, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2018.

[3] K. Chandrasekaran, *Essentials of cloud computing*. Chapman and Hall/CRC, 2014.

[4] S. Kunal, A. Saha, and R. Amin, "An overview of cloud-fog computing: Architectures, applications with security challenges," *Security and Privacy*, vol. 2, no. 4, p. e72, 2019.

[5] J. K. Z. N. C. T. U. Chuck Byers (IIC staff), Ron Zahavi (Microsoft), "The edge computing advantage," 2019.

[6] D. Jiang, "The construction of smart city information system based on the internet of things and cloud computing," *Computer Communications*, vol. 150, pp. 158–166, 2020.

[7] T. Xia, W. Zhang, W. Chiu, and C. Jing, "Using cloud computing integrated architecture to improve delivery committed rate in smart manufacturing," *Enterprise Information Systems*, pp. 1–20, 2020.

[8] H. Saini, A. Upadhyaya, and M. K. Khandelwal, "Benefits of cloud computing for business enterprises: A review," *Available at SSRN 3463631*, 2019.

[9] M. Szczerba, M. S. Wiewiórka, M. J. Okoniewski, and H. Rybiński, "Scalable cloud-based data analysis software systems for big data from next generation sequencing," in *Big Data Analysis: New Algorithms for a New Society*, pp. 263–283, Springer, 2016.

[10] B. Langmead and A. Nellore, "Cloud computing for genomic data analysis and collaboration," *Nature Reviews Genetics*, vol. 19, no. 4, p. 208, 2018.

[11] M. Doheir, A. S. H. Basari, B. Hussin, N. M. Yaacob, S. S. A. Al-Shami, *et al.*, "The new conceptual cloud computing modelling for improving healthcare management in health organizations," *International Journal of Advanced Science and Technology 351-362*, vol. 28, no. 1, pp. 351–362, 2019.

[12] Y. Liu, L. Zhang, Y. Yang, L. Zhou, L. Ren, F. Wang, R. Liu, Z. Pang, and M. J. Deen, "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE Access*, vol. 7, pp. 49088–49101, 2019.

[13] S. Prathiba and S. Sowvarnica, "Survey of failures and fault tolerance in cloud," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pp. 169–172, Feb 2017.

[14] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, 2018.

[15] T. Islam and D. Manivannan, "Facs: Toward a fault-tolerant cloud scheduler leveraging long short-term memory network," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 1–6, IEEE, 2019.

[16] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *Journal of Network and Computer Applications*, vol. 74, pp. 66–85, 2016.

[17] M. Alshayeji, M. Al-Rousan, E. Yossef, and H. Ellethy, "A study on fault tolerance mechanisms in cloud computing," *Int J Comput Electr Eng*, vol. 10, pp. 574–538, 2018.

[18] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: deep learning for system health prediction of lead times to failure in hpc," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 40–51, 2018.

[19] H. Agarwal and A. Sharma, "A comprehensive survey of fault tolerance techniques in cloud computing," in *2015 International Conference on Computing and Network Communications (CoCoNet)*, pp. 408–413, IEEE, 2015.

[20] S. Talwani and I. Chana, "Fault tolerance techniques for scientific applications in cloud," in *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pp. 1–5, IEEE, 2017.

[21] G. Louppe, "Understanding random forests: From theory to practice," *arXiv preprint arXiv:1407.7502v3*, 2015.

[22] T. Ropinski, D. Archambault, M. Chen, R. Maciejewski, K. Mueller, A. Telea, and M. Wattenberg, "How do recent machine learning advances impact the data visualization research agenda?," *IEEE VIS Panel. Phoenix*, 2017.

[23] D. Ramachandram and G. W. Taylor, "Deep multimodal learning: A survey on recent advances and trends," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 96–108, 2017.

[24] G. Öttl, "Analyze prometheus metrics like a data scientist." `https://promcon.io/2017-munich/slides/analyze-prometheus-metrics-like-a-data-scientist.pdf`, 2020. [Online; accessed 26-March-2020].

[25] E. Dubrova, *Fault-tolerant design*. Springer, 2013.

[26] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[27] S. Ghosh, *Distributed Systems, 2nd Edition*. Chapman and Hall/CRC, 2014.

[28] K. Kim, "Issues insufficiently resolved in century 20 in the fault-tolerant distributed computing field," in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, pp. 106–115, IEEE, 2000.

[29] M. Hasan and M. S. Goraya, "Fault tolerance in cloud computing environment: A systematic survey," *Computers in Industry*, vol. 99, pp. 156–172, 2018.

[30] J. Meenakumari, "Virtual machine (vm) earlier failure prediction algorithm," *International Journal of Applied Engineering Research*, vol. 12, no. 20, pp. 9285–9289, 2017.

[31] A. Alkasem, H. Liu, D. Zuo, and B. Algarash, "Cloud computing: a model construct of real-time monitoring for big dataset analytics using apache spark," in *Journal of Physics: Conference Series*, vol. 933, p. 012018, IOP Publishing, 2018.

[32] G. M. Qasem and B. Madhu, "Proactive fault tolerance in cloud data centers for performance efficiency," *Int J Pure Appl Math*, vol. 117, no. 22, pp. 325–329, 2017.

[33] D. Liu, B. Wang, P. Li, R. J. Stones, T. G. Marbach, G. Wang, X. Liu, and Z. Li, "Predicting hard drive failures for cloud storage systems," in *Algorithms and Architectures for Parallel Processing* (S. Wen, A. Zomaya, and L. T. Yang, eds.), (Cham), pp. 373–388, Springer International Publishing, 2020.

[34] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised hpc system and application," *Cluster Computing*, vol. 22, no. 2, pp. 471–485, 2019.

[35] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2009.

[36] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou, *et al.*, "Improving service availability of cloud systems by predicting disk error," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 481–494, 2018.

[37] B. Lai, "Predicting server failures with machine learning," tech. rep., SLAC National Accelerator Lab., Menlo Park, CA (United States), 2018.

[38] A. Fadaei Tehrani and F. Safi-Esfahani, "A threshold sensitive failure prediction method using support vector machine," *Multiagent and Grid Systems*, vol. 13, no. 2, pp. 97–111, 2017.

[39] A. Chigurupati, R. Thibaux, and N. Lassar, "Predicting hardware failure using machine learning," in *2016 Annual Reliability and Maintainability Symposium (RAMS)*, pp. 1–6, IEEE, 2016.

[40] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel, "A practical approach to hard disk failure prediction in cloud platforms: Big data model for failure management in datacenters," in *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 105–116, March 2016.

[41] F. D. d. S. Lima, G. M. R. Amaral, L. G. d. M. Leite, J. P. P. Gomes, and J. d. C. Machado, "Predicting failures in hard drives with lstm networks," in *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 222–227, 2017.

[42] H. Adamu, B. Mohammed, A. B. Maina, A. Cullen, H. Ugail, and I. Awan, "An approach to failure prediction in a cloud based environment," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 191–197, Aug 2017.

[43] J. Shetty, R. Sajjan, and G. Shobha, "Task resource usage analysis and failure prediction in cloud," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 342–348, IEEE, 2019.

[44] M. Jassas and Q. H. Mahmoud, "Failure analysis and characterization of scheduling jobs in google cluster trace," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pp. 3102–3107, IEEE, 2018.

[45] A. Bala and I. Chana, "Intelligent failure prediction models for scientific workflows," *Expert Systems with Applications*, vol. 42, no. 3, pp. 980–989, 2015.

[46] A. Rosa, L. Y. Chen, and W. Binder, "Predicting and mitigating jobs failures in big data clusters," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 221–230, IEEE, 2015.

[47] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1111–1116, IEEE, 2019.

[48] B. Brazil, *Prometheus: Up Running*. O'Reilly Media, Inc., 2018.

[49] Y. Kim, J. Woo, J. Lee, and J. S. Shin, "High-quality data collection for machine learning using block chain," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 23, no. 1, pp. 13–19, 2019.

[50] Z. Obermeyer and E. J. Emanuel, "Predicting the future—big data, machine learning, and clinical medicine," *The New England journal of medicine*, vol. 375, no. 13, p. 1216, 2016.

[51] A. R. Mashhadi, W. Cade, and S. Behdad, "Moving towards real-time data-driven quality monitoring: A case study of hard disk drives," *Procedia Manufacturing*, vol. 26, pp. 1107–1115, 2018.

[52] H. Yang, "Data preprocessing," 2018.

[53] G. M. Qasem and B. Madhu, "A classification approach for proactive fault tolerance in cloud data centers," *International Journal of Applied Engineering Research*, vol. 13, no. 22, pp. 15762–15765, 2018.

[54] D. Paper, *Hands-on Scikit-Learn for Machine Learning Applications: Data Science Fundamentals with Python*. Apress, 2019.

[55] F. Hutter, J. Lücke, and L. Schmidt-Thieme, "Beyond manual tuning of hyperparameters," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 329–337, 2015.

[56] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," *arXiv preprint arXiv:1502.02127*, 2015.

[57] M. Feurer and F. Hutter, *Hyperparameter Optimization*, pp. 3–33. Cham: Springer International Publishing, 2019.

[58] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826–830, 2017.

[59] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Springer, 2019.

[60] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

[61] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in *ICML workshop on AutoML*, vol. 9, Citeseer, 2014.

[62] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[63] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[64] G. Ranjan, A. K. Verma, and S. Radhika, "K-nearest neighbors and grid search cv based real time fault monitoring system for industries," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*, pp. 1–5, IEEE, 2019.

[65] D. Berrar, "Cross-validation," *Encyclopedia of Bioinformatics and Computational Biology*, pp. 542–545, 2019.

[66] C.-V. P. REFAEILZADEH, L. Tang, and L. HUAN, "Arizona state university encyclopedia of database systems," 2009.

[67] T.-T. Wong, "Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation," *Pattern Recognition*, vol. 48, no. 9, pp. 2839–2846, 2015.

[68] E. Scornet, G. Biau, J.-P. Vert, *et al.*, "Consistency of random forests," *The Annals of Statistics*, vol. 43, no. 4, pp. 1716–1741, 2015.

[69] A. Sarica, A. Cerasa, and A. Quattrone, "Random forest algorithm for the classification of neuroimaging data in alzheimer's disease: A systematic review," *Frontiers in aging neuroscience*, vol. 9, p. 329, 2017.

[70] A. D. Kulkarni and B. Lowe, "Random forest algorithm for land cover classification," 2016.

[71] L. Khaidem, S. Saha, and S. R. Dey, "Predicting the direction of stock market prices using random forest," *arXiv preprint arXiv:1605.00003*, 2016.

[72] R. Ramo and E. Chuvieco, "Developing a random forest algorithm for modis global burned area classification," *Remote Sensing*, vol. 9, no. 11, p. 1193, 2017.

[73] J. Kim and S. Grunwald, "Assessment of carbon stocks in the topsoil using random forest and remote sensing images," *Journal of environmental quality*, vol. 45, no. 6, pp. 1910–1918, 2016.

[74] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[75] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[76] P. Dangeti, *Statistics for Machine Learning*. Packt Publishing, 2017.

[77] P. D. A. Y. Allen Yu, Claire Chung, *Numerical Computing with Python*. Packt Publishing, 2018.

[78] V. Athanasiou and M. Maragoudakis, "A novel, gradient boosting framework for sentiment analysis in languages where nlp resources are not plentiful: a case study for modern greek," *Algorithms*, vol. 10, no. 1, p. 34, 2017.

[79] N. Chakrabarty, T. Kundu, S. Dandapat, A. Sarkar, and D. K. Kole, "Flight arrival delay prediction using gradient boosting classifier," in *Emerging Technologies in Data Mining and Information Security*, pp. 651–659, Springer, 2019.

[80] D. S. Jain, S. R. Gupte, and R. Aduri, "A data driven model for predicting rna-protein interactions based on gradient boosting machine," *Scientific reports*, vol. 8, no. 1, pp. 1–10, 2018.

[81] X. Wang, G. Gong, and N. Li, "Automated recognition of epileptic eeg states using a combination of symlet wavelet processing, gradient boosting machine, and grid search optimizer," *Sensors*, vol. 19, no. 2, p. 219, 2019.

[82] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[83] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, *et al.*, "Exascale deep learning for climate analytics," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 649–660, IEEE, 2018.

[84] A. G. Sujit Pal, *Deep Learning with Keras*. Packt Publishing, 2017.

[85] S. Kostadinov, *Recurrent Neural Networks with Python Quick Start Guide*. Packt Publishing, 2018.

[86] S. G. B. Armando Fandango, Rajalingappaa, *Python: Advanced Guide to Artificial Intelligence*. Packt Publishing, 2018.

[87] S. Ravichandiran, *Hands-On Deep Learning Algorithms with Python*. Packt Publishing, 2019.

[88] I. Vasilev, *Advanced Deep Learning with Python*. Packt Publishing, 2019.

[89] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[90] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[91] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[92] J. D. Smith, *Hands-On Artificial Intelligence for Beginners*. Packt Publishing, 2018.

[93] N. P. G. T. L. A. Iffat Zafar, Richard Burton, *Hands-On Convolutional Neural Networks with TensorFlow*. Packt Publishing, 2018.

[94] B. Rai, "Feature selection and predictive modeling of housing data using random forest," *International Journal of Industrial and Systems Engineering*, vol. 11, no. 4, p. 5, 2017.

[95] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, pp. 1019–1027, 2016.

[96] H. E.-A. M. Hamdy, *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*. Apress, 2019.

[97] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint arXiv:1811.12808*, 2018.

[98] J. Akosa, "Predictive accuracy: a misleading performance measure for highly imbalanced data," in *Proceedings of the SAS Global Forum*, pp. 2–5, 2017.

[99] M. Hossin and M. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, p. 1, 2015.

[100] G. Bonaccorso, *Machine Learning Algorithms*. Packt Publishing, 2018.

[101] J. Capellman, *Hands-On Machine Learning with ML.NET*. Packt Publishing, 2020.

[102] A. F. Ankit Jain, Amita Kapoor, *TensorFlow Machine Learning Projects*. Packt Publishing, 2019.

[103] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

List of Tables

List of Figures

Appendix

I. List of Acronyms

VM Virtual Machine

KNN K Nearest Neighbor

SVM Support Virtual Machine

RF Random Forest

CART Classification and Regression Trees

LDA Linear Discriminant Analysis

HPC High Performance Computing

SMART Self-Monitoring, Analysis and Reporting technology

NERSC National Energy Research Scientific Computing Center

CFDR Computer Failure Data Repository

LSTM Long Short-Term Memory

II. List of Notations

Notation	Meaning
λ	Hyperparameter
∂	Mathematical Derivative
γ	Output value
$\odot$	Element wise multiplication
$\hat{}$	Predicted

III. Gradient Boosting Calculation

In this section, we shall see in detail the application of the gradient boosting algorithm with an example data-set 6. We are going to use the simplified formulae. Stammer[37] explains simplification steps in detail and Friedman et. al provides details of the math involved in this algorithm [103], which can be referred for details. The details of the algorithms are also described in Section 3.6. We, therefore, in this, only focus on calculation. Further, we assumed the learning rate of 0.1.

As stated in the algorithm we first have to calculate the initial prediction and assign it. The calculation of the prediction is based on the log odds, so we calculate log odd and obtain its value as 0.69. And since it is our first step, F_0(x) also becomes the same i.e. 0.69.

$$logodd = log_e(2/1) = 0.69 \tag{33}$$

Next, we calculate the initial prediction based on the log odds value, and using the calculated initial prediction probability $P_0 = 0.67$, the residual shown in Table 11.

$$predictionprobability(P) = \frac{e^{logodd}}{1 + e^{logodd}} \tag{34}$$

Server	smart_198	smart_3	Fail?	Residual
A	19	9	Yes	0.335
B	100	98	No	-0.665
B	61	9	Yes	0.335

Table 11. Residual

We then follow the fit-regression step for constructing the tree and mark the label Figure 24. Afterward, we calculate the γ value for the respective label. γ can be calculated as the sum of residual divided by sum of P(1-P).

$$\gamma_{11} = \frac{0.335}{(1 - .67) \times 0.67} = 1.51 \tag{35}$$

$$\gamma_{12} = \frac{-0.665 + 0.335}{(1 - .67) \times 0.67 + (1 - .67) \times 0.67} = -0.7462 \tag{36}$$

[37]Gradient Boost Classification Details StatQuest

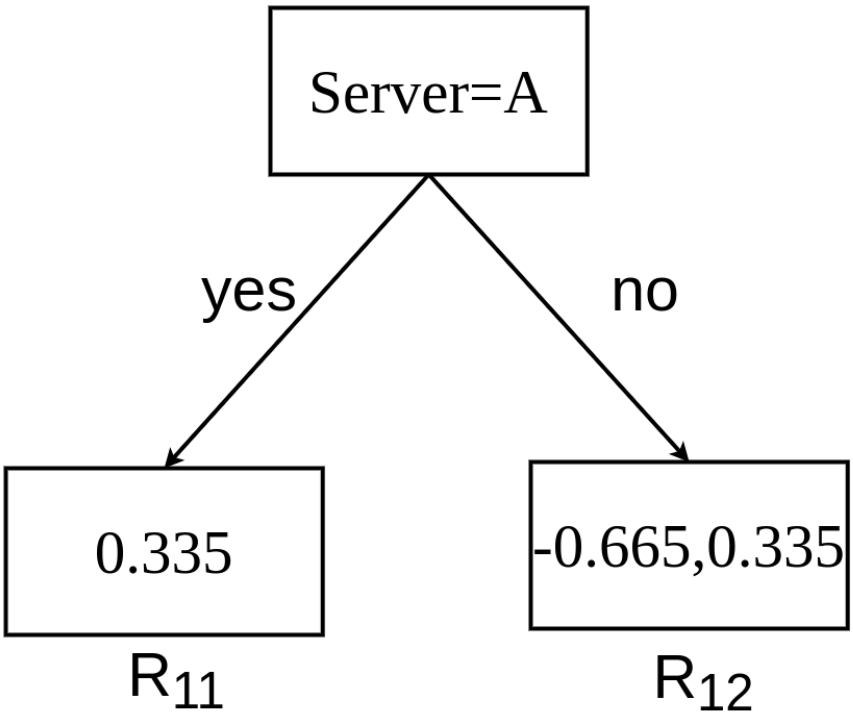

Figure 24. Fitting regression tree

We run our data through tree to find the appropriate value of γ and calculate the new log-odds F_1(x) = F_0(x)+learning rate $\times$ γ.

Server	**smart_198**	**smart_3**	**Fail?**	**Residual**	**Logodd1**	**P1**
A	19	9	Yes	0.335	$0.69 + 0.1 \times .51 = 0.84$	0.6984
B	100	98	No	-0.665	$0.69 + 0.1 \times 0.7462 = 0.61538$	0.6491
B	61	9	Yes	0.335	$0.69 + 0.1 \times 0.7462 = 0.61538$	0.6491

Table 12. New Log odds and prediction probability for Iteration 1

For the next iteration, we follow the same. First, we begin with the calculation of the

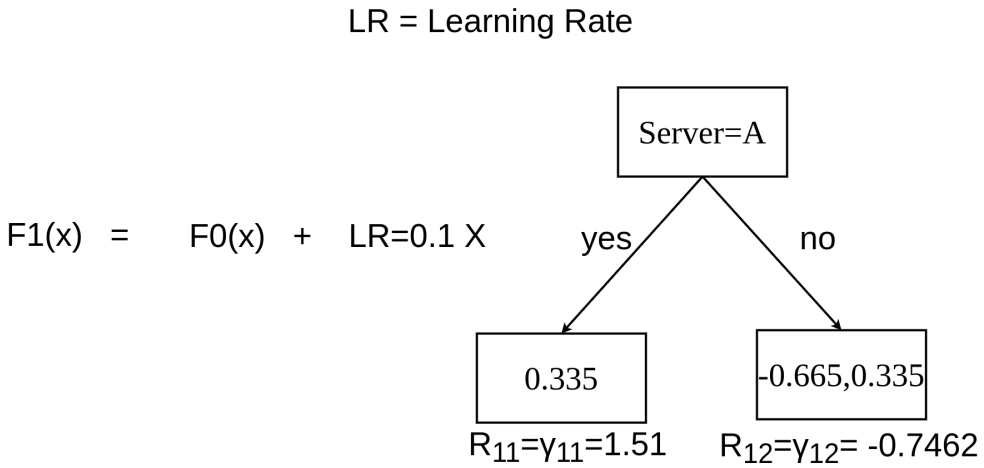

Figure 25. F_1(x) for new log odd calculation

residual which is observed minus predicted. We already have the previous prediction from iteration 1 and the observed value provided in the original data-set. Upon calculation, we get the following residual Table 13.

Residual
0.3016
-0.6491
0.3509

Table 13. New Residual for Iteration 2

We follow the repeated procedure of fitting the regression tree 26, followed by a calculation of γ. The calculation yields 1.4318 and -0.6546 respectively for γ_{11} and γ_{12} . Further using the constructed tree Figure 26 and the calculated γ, we calculate new log-odds again running though this new tree and using the previously calculated F_1(x) and the respective prediction. This new calculation of log-odd follows F_2(x) shown in Figure 14. This process of calculation and update of the value is repeated until the stopping criteria defined by hyperparameter is reached.

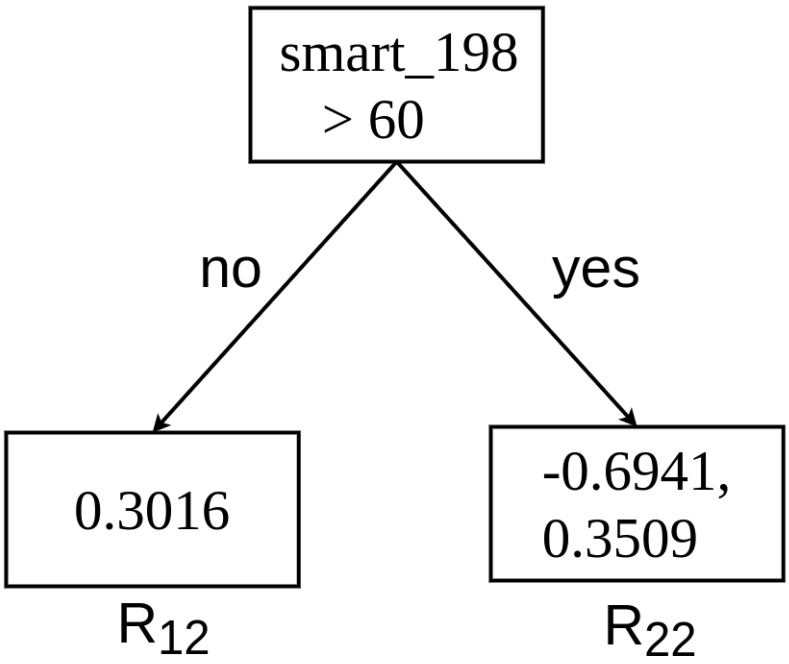

Figure 26. F_2(x) for log odd calculation for iteration 2

IV. Source Code

The source code and trained model for this study is located in repository `https://bitbucket.org/tekrajchhetri/cloud-resources-failure-prediction/`.The access to the repository could be granted upon sending an email to tekrajchhetri@gmail.com.

7 Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Tek Raj Chhetri**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Towards AI for cloud services reliability using combined metrics,

   supervised by **Prof. Satish Narayana, Dr. Chinmaya Dehury and Artjom Lind**.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Tek Raj Chhetri
15/05/2020