UNIVERSITY OF THESSALY

ELECTRICAL AND COMPUTER ENGINEERING

RESEARCH ON AND IMPLEMENTATION OF MACHINE LEARNING ALGORITHMS FOR FAILURE PREDICTION IN THE OPERATION OF THE RAFT CONSENSUS ALGORITHM

Diploma Thesis

Nikolaos Christogiannis

Supervisor: Athanasios Korakis

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΛΕΤΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΑΛΓΟΡΙΘΜΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΠΡΟΒΛΕΨΗ ΑΠΡΟΣΔΟΚΗΤΩΝ ΜΕΤΑΒΟΛΩΝ ΣΤΗ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ ΣΥΝΑΙΝΕΣΗΣ RAFT

Διπλωματική Εργασία

Νικόλαος Χριστογιάννης

Επιβλέπων: Αθανάσιος Κοράκης

Βόλος 2021

Approved by the Examination Committee:

Supervisor	**Athanasios Korakis** Associate professor, Department of Electrical and Computer Engineering, University of Thessaly (Volos)
Member	**Antonios Argyriou** Associate professor, Department of Electrical and Computer Engineering, University of Thessaly (Volos)
Member	**Dimitrios Bargiotas** Associate professor, Department of Electrical and Computer Engineering, University of Thessaly (Volos)

Approval Date: 22-9-2021

Acknowledgements

I would like to thank my supervisor Athanasios Korakis, Associate Professor in the university of Thessaly as he entrusted me with this thesis and provided me all I needed to complete this project. Additionally, I would like to profoundly thank Konstantinos Houmas, a post-doctoral researcher at NITLAB for his immense support in this project, with his knowledge and experience he guided me many times throughout this thesis.

Furthermore, I would like to thank my best friends Aris, Christos and Xrysostomos whom I have known since childhood for their immeasurable support for anything and everything I do and for being there whenever I needed them. Lastly, I would like to thank my parents, my sister as well as my grandparents for their unconditional love and support and their wise teachings…without which this thesis would not be possible. All of these people share a piece of the success I will ever experience.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών
ΝΙΚΟΛΑΟΣ ΧΡΙΣΤΟΓΙΑΝΝΗΣ
22/9/2021

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The 11 points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

NIKOLAOS CHRISTOGIANNIS

22/9/2021

ABSTRACT

The aim of this diploma thesis is to offer mechanisms for predicting the cause of a failure in a cluster of distributed state machines by using machine learning algorithms. The algorithm used to distribute those state machines is the raft consensus algorithm and the machine learning algorithms used and compared in this thesis are Decision Tree, Random Forest, Support Vector Machine (SVM), Adaptive Boosting and Extreme Gradient Boosting. The cluster was simulated using Mininet and ETCD instances and python scripts were developed to automate the process of causing random failures in the cluster and capturing the messages of the communicating members as well as to clean and prepare the dataset for the machine learning algorithms. Conclusively, after experimental simulations were run the machine learning algorithms showed promising results and it is estimated that this technique could be improved and then used in further research on the subject for real life testing and improvements on orchestration systems such as Kubernetes.

ΠΕΡΙΛΗΨΗ

Ο στόχος της διπλωματικής είναι να παρέχει μηχανισμούς για την πρόβλεψη της αιτίας μιας αποτυχίας σε συστάδα κατανεμημένων μηχανών κατάστασης με την χρήση αλγορίθμων μηχανικής μάθησης. Ο αλγόριθμος που χρησιμοποιείται από την συστάδα είναι ο αλγόριθμος ομοφωνίας Raft και οι αλγόριθμοι μηχανικής μάθησης που χρησιμοποιούνται και συγκρίνονται στο κομμάτι της πρόβλεψης είναι οι Decision Tree, Random Forest, Support Vector Machine (SVM), Adaptive Boosting και Extreme Gradient Boosting. Η συστάδα προσομοιώνεται με την χρήση Mininet και ETCD και οι διαδικασίες πρόκλησης τυχαίων αποτυχιών και της λήψης των μηνυμάτων επικοινωνίας μεταξύ των μηχανών αυτοματοποιήθηκαν μέσω προγραμμάτων python όπως και για την εκκαθάριση και την προετοιμασία του dataset. Καταληκτικά, μετά από πειραματικές προσομοιώσεις οι αλγόριθμοι μηχανικής μάθησης έδειξαν πολύ θετικά αποτελέσματα για την εξέλιξη και βελτίωση της τεχνικής γεγονός που ανοίγει προοπτικές για περαιτέρω έρευνα πάνω σε πειραματισμούς πραγματικού υλικού εκτός προσομοιώσεων και σε συστήματα διαχείρισης συστάδων όπως το Kubernetes.

TABLE OF CONTENTS

Figure Table

Chapter 1: Introduction

1.1 Problem Statement and Motivation

In a world of great technological advancement computers operate in every single aspect of everyday life for the betterment of society and palliation of bureaucracy burdens. Everyday computers and servers worldwide handle billions of user requests for both entertainment and work facets of life as well as government related responsibilities and services but given the scale of the users and the services provided no-one computer could possibly control all the traffic by offering both reliability and fast processing. The most widely used method to solve this issue is to use multiple computers to substitute for the one in order to load balance the requests and improve system-network latency and reliability. However, this solution ought to be seamless for the user and should not concern anyone beyond the developers and system administrators, which means that this process has to happen dynamically in real-time and usually without human-input. It becomes apparent that if thousands of requests flood a server in seconds or minutes, the load balancing acts should be performed by machines rather than people, which creates the need for software cluster management and monitoring. Currently, there are multiple software solutions that manage clusters and deal with the faults that may arise in their operation with some of them being Kubernetes, Docker Swarm, Microsoft Cluster Server and more. The way that software deal with cluster-failure recovery is different but the case of Kubernetes, which uses ETCD and the Raft protocol, will be of concern in this thesis. The aim of this thesis is to provide tools to predict the type of failure that a cluster with ETCD and Raft endured so that it could be possible to influence the way that the cluster recovers from that failure in a more optimized way rather than randomly as it happens now, and it is considered that this could improve cluster stability and performance. The predictive tools used are machine learning algorithms and more specifically: Decision Tree, Random Forest, Support Vector Machine (SVM), Adaptive Boosting και Extreme Boosting, which will be analyzed in the following sections below.

1.2 Thesis Structure

The structure of the thesis will be described in the following paragraphs so that each chapter has a clear distinction and purpose, therefore the reader is directed to read a brief overview of the chapters and/or use the table of contents for more direct navigation.

Chapter 2 will explain the theoretical background required so that the problem at hand is clear, and the purpose of the thesis is stated but more specifically in 2.2.1 and 2.2.2 there will be an introduction to computer clusters and data orchestration respectively. Additionally, in 2.2.3 and 2.2.4 the need for orchestration tools is discussed and the capabilities they offer are described. Moreover, in 2.2.5 the Raft algorithm is briefly explained and 2.3 it describes of the scripts that are utilized in this thesis so that the processes of extracting the data from the clusters and creating the datasets are automated with little interaction.

Chapter 3 briefly explains the theoretical background of the machine learning algorithms used; however, it will not dwell too much in the mathematics side of the algorithms as it is not of particular interest in this thesis. Also, the training results of the different algorithms are presented with a comparison between them so that a potential better algorithm could stand out. All experiments were run in simulations of three cluster-nodes with the same exact datasets so that a valuable comparison could be possible.

Finally, the conclusions of the thesis are being drawn as to which methods worked the best and how this could be researched and utilized further for influencing the Raft elections and optimizing the election results.

Chapter 2: Theoretical Background, Tools and Scripts

2.1 Chapter Overview

A brief overview of the theoretical background required will be explained so that computer clusters and Raft are clear enough subjects for the reader, so that they understand the following sections as well as the existing protocols of ETCD/Raft and why there is room for improvement in some areas of operation. Furthermore, the scripting process will be explained, as python scripts were developed in order to automate the process of extracting the data from the operation of ETCD/Raft and forming the dataset in a way best suited for the algorithms. Conclusively, in the following sections there will be mentions of the information and these tools mentioned in this chapter, so they are deemed essential to the overall understanding of the thesis.

2.2 Theoretical Background

2.2.1 Computer Clusters

Modern computers have given the everyday user the ability to study and work from home, deal with government related activities, entertain themselves, socialize with other people as well as shop and game. Most of these aspects of life have started attracting more and more people in the digital world and with the use of very powerful computers the case for thousands, millions and potentially billions of everyday users for these services has been rendered considerably more realistic. However, even the most powerful of the world`s supercomputers could not handle all the traffic that flows through the worldwide web every day, therefore the need for multiple computers working together on the same tasks has risen. Computer clusters are a set of computers usually used to replace a single computer in order to improve stability and speed in data-processing and they are usually utilized by businesses and web services so that the content they provide is always available regardless of the traffic they get or the hardware limitations (memory etc.). As mentioned before, one of the key benefits of using clustered machines is speed in processing because of the much greater hardware power, which means that more users can concurrently use the services with less latency therefore making the service much more responsive and easier to use without crashes or delays. Almost every cluster utilizes this benefit for its service because of the needs of modern online businesses and services with some very widely used ones being social-

networks and video streaming platforms etc. In these examples not only is the user-base very big (millions or even billions) which enhances the need for more hardware resources, but the users demand an instantaneous response in their requests because messages and emails can not take hours or days to be sent and videos can not lag and freeze every minute or take hours and days to load. Furthermore, another key benefit of the clustered architecture is the ability to provide the users uninterrupted services in case of hardware or software failures in the machines or also known as failover capabilities. Failover enhances the user experience and in turn the service: if a node fails the service is still online and running with every other node working properly and the user gets redirected to use another node in the cluster. Unlike clusters, computer failures in non-cluster implementations could lead to the crash of the whole service making it impossible to use, which means it is a bad business practice for servers with large amounts of traffic and low latency demands. There is an issue with this architecture however, the data between the server nodes must be frequently updated so that they store the same data, and the end users always get the most up to date information, this is a very important issue that will be discussed about further in 2.2.2.

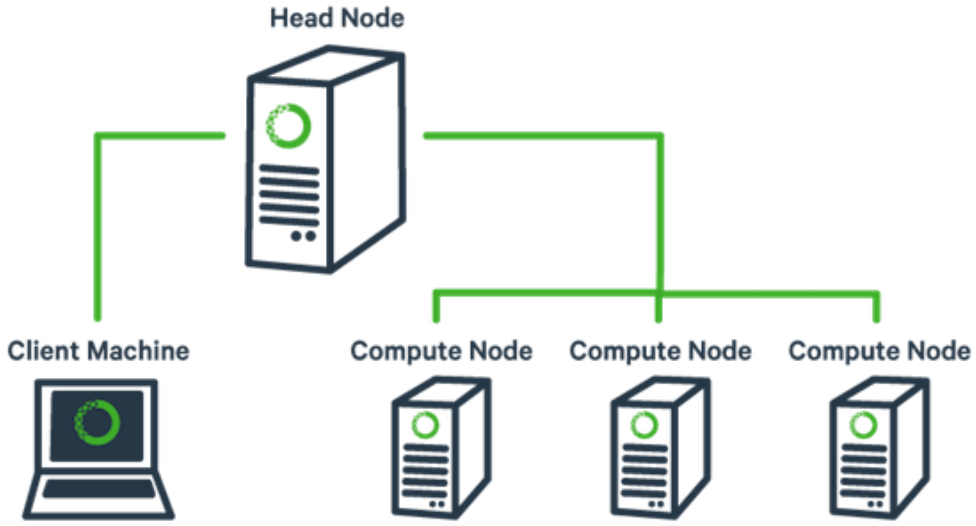

Figure 2.1: A graphic diagram of what a clustered architecture looks like. The head node is responsible of tasking the other "slave" nodes of the work, and it is its responsibility to not overwhelm any-one node as well as keep track of nodes that are down. Graph taken from https://docs.anaconda.com/anaconda-cluster/

2.2.2 Data Storing and Synching in Clusters

As mentioned before clusters are used by high-end users that require their services to be distributed in multiple computers both for low-latency and higher reliability and accessibility. However, the cluster offers reliability not only because it can run after node failures but because it can store the data in multiple physical memories so that the

information is secure in case of hardware failures or disasters (fires, floods etc.). Although simple users usually do not require such mechanisms for data security, the reader might be familiar with services like Google Drive, Microsoft OneDrive, Dropbox etc. which are cloud products that offer data redundancy in case of data loss or hardware failure (even stolen or lost devices) which offer likewise capabilities. As a sidenote it is recommended to the reader that they perform frequent data backups in different physical memories (if not in cloud services) because the value of personal data (photos, videos) is invaluable and can only be measured when lost. Whereas simple users might not feel the need to back up their data (not recommended) such practices can not be followed by enterprises or important services with crucial information like health departments with electronic health records

Figure 2.2: The IBM Blue Gene/P supercomputer installation at the Argonne Leadership Angela Yang Computing Facility located in the Argonne National Laboratory, in Lemont, Illinois, USA. All modern supercomputers utilize the power of clustered machines. Taken from https://commons.wikimedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg

or treasury departments that keep all the taxed individuals in check, or even banks that store individuals bank accounts instead of in written form in books. Consequently, data storing is extremely important in clusters, however an issue occurs if multiple machines work with the same data but in isolated physical memories, the case of out of sync data. The problem of data synching between cluster nodes might not be an obvious one, however if one delves a little deeper in what kind of problems users can encounter when served by different nodes it

can be very enlightening. An example of this problem could be a failure in a bank server, if the data of the bank transactions were not updated in another system, all transactions that occurred in that timespan would not have been valid and would have to be performed again which is very inconvenient and very unreliable. Another example is a cloud storage service that has a crashed node, if that node had not updated at least one of the other nodes with the new information, the data would be lost, and the user would lose track of all changes. Not only that, but to add to that last example, the reader can imagine a single file stored on multiple severs that many users have access to, something that has become very common in this age. Would the users be in completely different geographical locations, the servers that serve them will probably be very different for each user, however the data has to be synchronized. Had those users made conflicting changes to the file, then some user`s edits would be lost forever, or the system would not know which version to offer to the clients the next time they request the file. The way this is solved in modern clusters is by using software that manages the cluster and controls how the data is updated and how often, therefore the process of backing up the data is automated, and the system is safer in case of failures. It is important however to note that even when these programs are used, the data is not completely secured in case of hardware or cluster failures and further methods are utilized for more reliability and redundancy.

Figure 2.3:A Microsoft cloud computing center in Quincy. This is what a modern data storage facility looks like where millions of gigabytes are stored. Wash.Credit...Richard Duvall. Picture taken from https://www.nytimes.com/2017/01/23/insider/where-does-cloud-storage-really-reside-and-is-it-secure.html

2.2.3 Cluster Management and Orchestration Systems

Cluster management tools are software tools that help system administrators to monitor and configure computer clusters according to their designs and the real time performance metrics. The demand for cluster management tools is of great importance if one considers the need to frequently update and resolve software issues in clustered nodes simultaneously. Whereas this would not have been an issue if the cluster had a small number of members, modern clusters however could potentially support up to thousands of nodes as noted in [1] therefore the manual way of installing and updating the nodes is not viable. These tools are called orchestration tools because they orchestrate the clusters and automate their execution as well as do everything that is necessary to make failed nodes run again and restore the cluster to a healthy state. As a result of the great demand for such tools, there are several solutions that make the job much easier for the teams of system administrators, with some of them being open source and some of them being proprietary, but a few of the most widely

used ones are Kubernetes[1] (Google), Docker[2] (Swarm), Mesos[3] (Apache), Microsoft Cluster Server[4] and more. Kubernetes has gotten popular quickly in the orchestration tools space because it is open source, and it has been developed by Google in 2014 who have had experience with this technology for many years and are a pioneer in this space, hence it is a very competitive solution. Kubernetes offers a dashboard UI for developers to take advantage of and monitor the cluster in a visual way, as well as detect anomalies in the performance of the nodes. Amongst other capabilities, developers can monitor CPU/ RAM frequency and usage, the number of CPU cores that are in use as well as the amount of time they are running and their current status (alive or down) an example of which can be seen in figure 2.4 below.

Moreover, users can utilize other software or packages for monitoring the cluster as well as notification systems that warn the developers of imminent crashes or failures, even though many of the recovery tasks are being taken care of by the automated system itself. For example, a couple of the more well-known monitoring software are Prometheus[5] and Grafana[6] which can be integrated into Kubernetes and provide the important metrics of the cluster as well as issue warnings and keep cluster logs. Even though these tools are not going to be of particular importance for this thesis, their existence and their use make clusters much easier to handle and operate and they are considered important for the understanding of the subject at hand.

2.2.4 Kubernetes and etcd

As previously mentioned in 2.2.3 above Kubernetes is an orchestration tool that helps developers and system administrators keep a cluster healthy and the service it provides uninterrupted. Additionally, a lot of features it comes with have been discussed with the more relevant ones being self-healing, data orchestration and load-balancing, but all the other ones are equally as important. Self-healing is a capability Kubernetes offers with which it can kill processes in nodes that are misbehaving or have crashed (are unresponsive) and can restart them while taking them off the list of healthy nodes until they are completely responsive and ready to serve requests again. In addition to that, the administrators can put

[1] https://kubernetes.io/
[2] https://www.docker.com/
[3] http://mesos.apache.org/
[4] https://www.microsoft.com
[5] https://prometheus.io/
[6] https://grafana.com/

in place custom health check standards so that a node or the cluster are considered healthy only when those standards have been met, which is especially useful in situations where stability and reliability is of great importance to the service. Furthermore, the data orchestration or data synching capabilities of the cluster, Kubernetes handles by utilizing etcd[7] which as noted in the official website is "a strongly consistent, distributed key-value store that provides a reliable way to store data that needs to be accessed by a distributed system or cluster of machines". More specifically, as mentioned in [2] etcd manages the configuration data, state data, and metadata for Kubernetes and the status of the cluster as well as the status of the processes of each node. To add to that, etcd is considered "fully replicated" which means that every node has its own database to store key-value pairs in, as well as consistent since every single read will provide the most recent write that has been performed to the data, but more on that in section 2.2.5. Another functionality of etcd is that it is Highly Available (HA) which means that there is no single point of failure for an etcd cluster, even if the master node goes offline. Lastly, etcd is fast but it is very dependent on network and hard drive speeds since it performs writes in memory, but it is also a secure platform because it is using Secure Sockets Layer (SSL) client security or Transport Layer Security (TLS) which are optional and highly advised client authentication techniques.

As previously described in section 2.2.2 as well as in this section, if there is no specific policy for the data-synchronization between the nodes there is a problem in what version of the data is more recent or more precise, which could be detrimental for most businesses and this kind of ambiguity can not be tolerated in a cluster if stability and reliability are to be expected. This is a problem that would be very difficult to fix if all nodes write to their databases and then the data must be merged, whereas etcd avoids this approach altogether and uses the Raft protocol for handling of "writes" in the logs and the database of the cluster. Raft is essential to etcd for making it a robust database that can withstand failures, but more on Raft will be discussed in section 2.2.5 below.

[7] https://etcd.io/

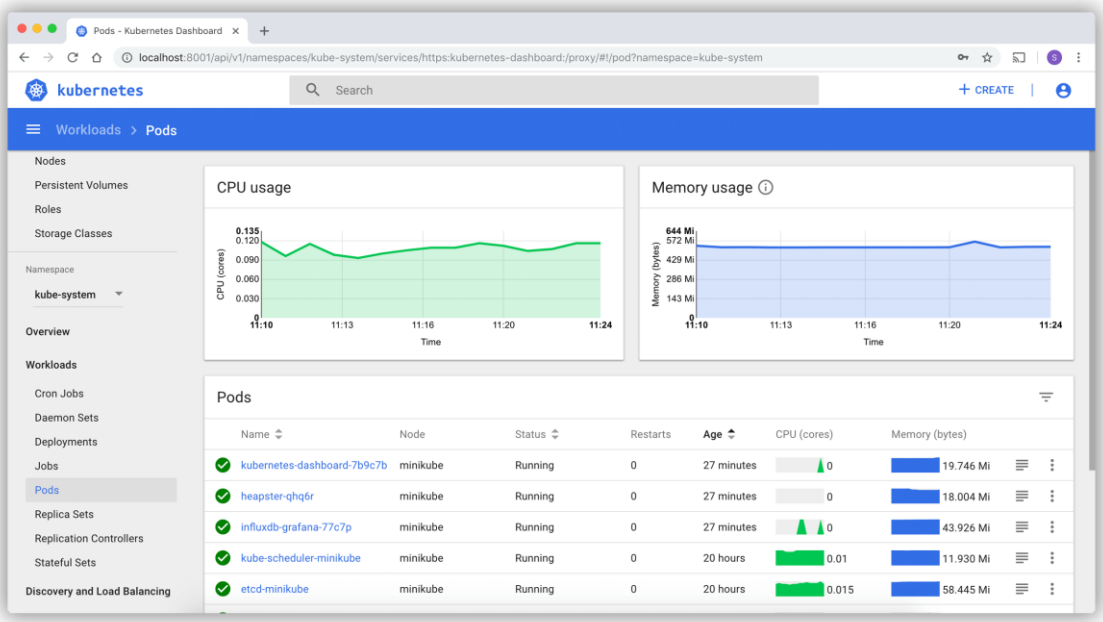

Figure 2.4: An example of the Kubernetes dashboard when monitoring the cluster diagnostics. Orchestration tools like this make cluster management much easier for the developers.

2.2.5 The Raft Algorithm

This section is a culmination of many of the sections above, not only because it is essentially the protocol this thesis aims to enhance but because it provides the reliability and consistency to etcd that was mentioned in 2.2.4. The Raft algorithm[8] as explained in [3] (developed based on the Paxos algorithm seen in [4]) is the way etcd manages to consistently provide the most recent "write" to the data when requested as opposed to an older or outdated version, the mechanism of which will be described in this section. Firstly, the Raft algorithm assigns to a node in the cluster the position of "leader" to designate that node as the more updated and informed one, with every other node in the cluster being a follower of that leader. The way this leader is picked is by random elections between the cluster nodes, with some node or nodes announcing their candidacy for leadership and expecting the vote responses from the other nodes. Additionally, every time a new election takes place (hence a leader change has occurred), a new term is established by the algorithm and if there is a consensus in the voting process then a new leader will have been elected and every other node would be a follower, however if there is no consensus then the elections start over and a new term is established. One way that it is attempted to avoid cases of leaderless elections (undecisive elections) is by requiring that the cluster have an odd number of nodes so that if all nodes are online there

[8] https://raft.github.io/

is always going to be a leader elected (obviously a node can not vote for more than one candidate). As was previously mentioned in the sections above, amongst other benefits, the cluster acts as a load balancer for the service that is being provided, and the leader has its followers take the incoming read requests so that it is not overloaded. In a case of a write request however, the leader gets the changed data and asks that every follower replicates that data and appends it to its log and requests a successful response of that replication, and only when and if that process updates the majority of the cluster with the new data it then replicates the data to its own local storage/log.

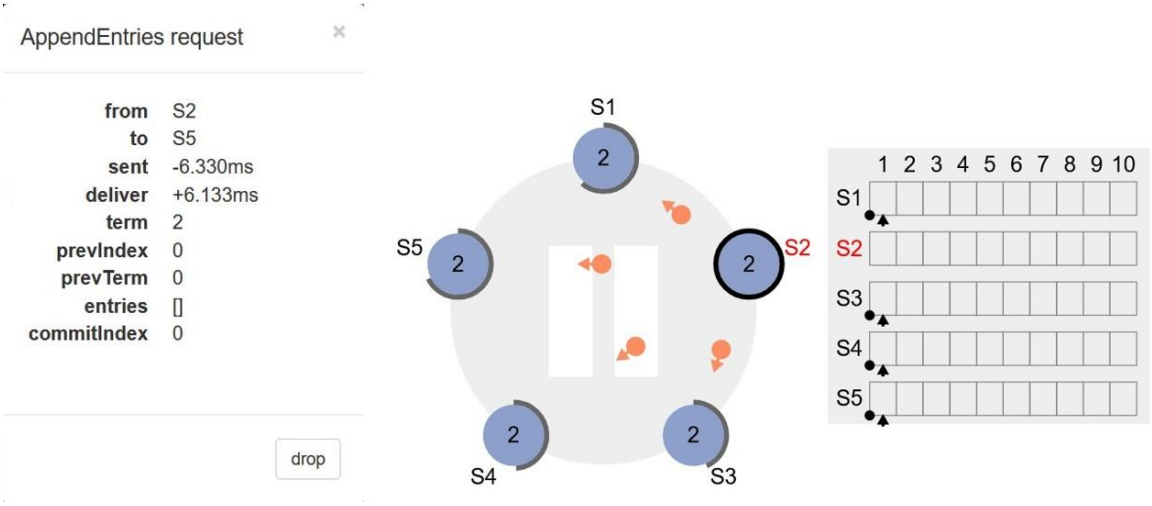

Figure 2.5: The (just-elected) leader S2 sends the append entries request in every other node in the cluster as the first operation of its tenure. This is very important so that everyone has the same updated information as the leader especially after a write in its database. Screenshot taken from https://raft.github.io/

Thus, since the leader has the most updated data at any single point in its database and the read requests end up to the followers, it is wise to consider that a "read" in a follower node could give misleading or false data back to a client. Because of that, Raft demands that every "read" that is requested from a follower node goes through the leader first, therefore if a new and updated version of the data is available then that will be the version that will be forwarded to the client. As a result of the above, if a follower crashes or goes offline the cluster will be safe since no data loss will be expected, even though stability might suffer before the problem is sorted. In contrast to a follower failure however, if a leader fails that is very important for the cluster, and that is when new elections are triggered. The type of failure is not necessarily the node going offline and crashing, but it could be a delay in a link (that connects two nodes) of the cluster being greater than the tolerated standard or a cut off link. The way this change in status is observed is because Raft has communicative messages between the cluster members to designate and differentiate cluster functions some of which is the election announcement, or the log replication success/request and the heartbeat

messages. These messages are very commonly used in the computer networks field, and they are used for nodes to know the status of their communication partners, an example of which is the routers in the worldwide web. The routers exchange “hello” messages to keep track of online and offline links so that they can re-route traffic in case of unreachable nodes as well as optimize the traffic balancing in case of dynamic routing algorithms. All these messages are very important in this thesis as they designate the failures in the cluster before a leader election has started, and their use is important to decide what failure has occurred. To add to that, the way they prove useful is because whenever the nodes send heartbeat messages, they expect a reply of acknowledgment of that message from the other node(s) as a way to indicate that they are still alive. If the timeout period of the heartbeat messages passes, then they request that new leader elections shall start as they assume that the leader is down and this is the key feature that the machine learning algorithms could tap into, the lack of replies in the communication between cluster-nodes.

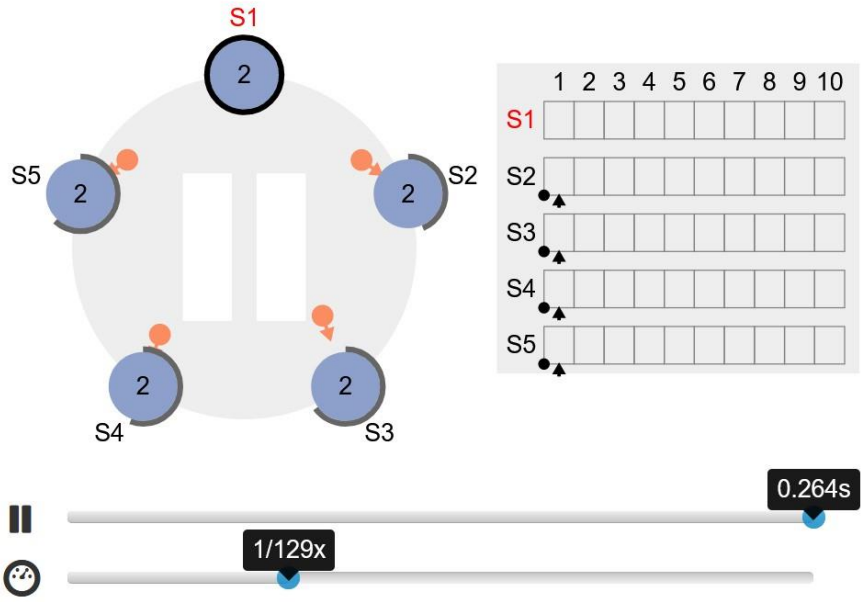

Figure 2.6: A Raft visualization with 5 servers. Server 1 is the leader and every other one is a follower. Screenshot taken from https://raft.github.io/

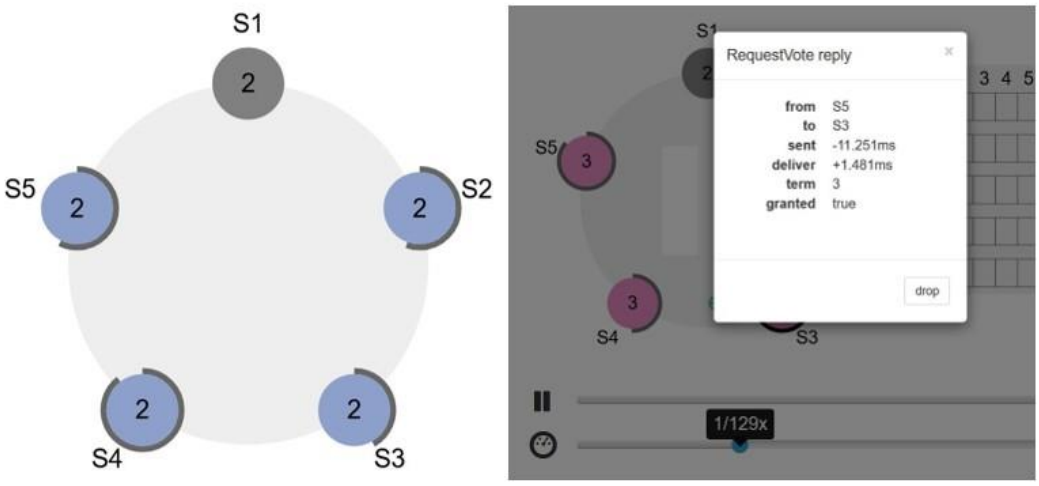

Figure 2.7: A) To the left it can be seen that server 1 has now been deactivated and every other follower is in timeout mode. When timeout ends, they send leadership vote requests. B) To the right a vote response can be seen after timeout has passed. Screenshot taken from https://raft.github.io/

2.3 Tools and Scripts

Introduction

The dataset for a machine learning application requires a great amount of data for the algorithm to train well enough, but the number of that data is not preset, and one can only experiment with the data in order to get the best results. In this thesis, there was experimentation with many dataset sizes, but the optimal size was considered at 1000 failures, which in turn means that there would have to be a way to automatically extract many messages from the cluster members. That is why scripts were developed which randomly create artificial failures based on which node the leader is and what type of failure would cause a new election and the end of a term as well as reform that data and repurpose it as a dataset. The nodes were simulated in Mininet, and the Raft algorithm was utilized by using the etcd-Raft Example[9]. Raft only handles clusters with an odd number of members so the experiments that will be of concern are with three nodes in the cluster.

2.3.1 Artificial Cluster Failures in Mininet

In the beginning, the script that had to be developed was the one that would create random failures in the cluster so that a dataset with hundreds or thousands of term changes could be created. A Mininet topology of the nodes interconnected with a switch has been created with

[9] https://github.com/etcd-io/etcd

hardcoded IP and MAC addresses so that they can be discoverable in the cluster. Because Mininet does not allow direct node to node connections, for the failures that require the link between two nodes to go offline, the proactively set switch table gets changed and the flows that connect the two nodes (that simulate the link that needs to go down) get removed from the switch table. The reasoning behind this logic is to let every other node be able to communicate with the node with the failed link so to simulate that only that link has a connectivity problem. In the beginning of the process, the script starts up the Mininet simulation described above and runs the Raft Example algorithm in every node while saving the output of those commands it in separate txt files, which it then scans to find leader changes between the nodes and attempts to synchronize all of them so that there is no error in the process. Once the leader has been identified properly then it randomly creates an artificial failure by either shutting down the link of the interconnected nodes or by shutting down the node-leader. It is essential that in every sample there is a leader change, so that the algorithms do not train on useless data and thus every time a failure is caused, the script either chooses a link that is connected to the leader or the leader itself as the failure. The failures and their metadata (terms and interfaces) are also being kept in files so that they can be utilized further in the future for the automation of the dataset making process. Moreover, the script starts tcpdump programs in order to capture the traffic between the nodes and saves it in pcap files which are then used in order to transform the exchanged Raft messages into humanly readable txt files.

At this point another script had to be developed that extracted the Raft algorithm message metadata from the TCP packets that were captured with TCPDUMP and saved that metadata in txt files so that they could be the dataset used in the machine learning algorithms. Every Raft message is converted into a single line of data that contains the message receiver, message sender, message term, message time, message vote response and type and the last heartbeat message received is also appended. This file is the type of input that the pre-processing script requires so that the dataset can be formed properly, and it can be exported in the appropriate form, more on which in section 2.3.2.

2.3.2 Dataset pre-processing

The script developed for the automation and formation of the dataset for this thesis was based on the messages exchanged and extracted from the Raft algorithm and the scripts developed in 2.2. The input of the script is a .txt file that contains one line for every message captured

during the run of the algorithm, and the metadata that was maintained were: receiver, sender, term, message time, message type of each packet and the last heartbeat message that was sent. The file containing all the artificial failures is used to coordinate and synchronize the txt file with the failures so that there is no mistake during the dataset formation process. The idea was that every bit of information described above would be a column in the dataset for the machine learning algorithms to train on, and every one of these columns would contain crucial information as to which link had failed based on the exchanged messages. Information like receiver and sender indicate the node of the cluster that either received or sent the message respectively, and term, time indicate the term and time that the message was sent at. The message type is an important indicator of what type of operation Raft is trying to enable, and since there are some signs that a new election might be carried out hence a failure of some kind has occurred in the cluster this field is very useful. The different message types are noted with different integer number but of greater importance are considered of being the heartbeat messages which just verify that nodes are alive. Consequently, if there are many heartbeat messages to one node from a node this could be an indication of a failed link whereas many heartbeats from many nodes could indicate that a node has crashed. The differentiation and prediction of the failures based on the previous messages is what machine learning is tasked to do in this thesis by training on the dataset of artificial failures. However, a valid concern would be, what is the optimal number of messages to look back to in order to decern what kind of failure has happened and after some experimenting the optimal number was found to be 20 messages before the first heartbeat message that appears to have a new term. Therefore, a new script had to be developed so that the txt file could be reformed, and a clean-up process could be possible. This script searches the dataset for a new term-heartbeat pair and captures the previous 20 messages of that heartbeat as a history for that failure and then it changes all of these messages to make their data in relative form. More specifically, time and terms are considered zero for the oldest “history” message and all the other messages are adapted relatively to that change with both negative and positive numbers, however message type. vote response as well as sender and receiver were not changed. In addition to that, these messages are reformed into a single row of the dataset and another field of data is added: the failure type. The failure type is taken from the file that has been kept from the operation of the scripts in 2.3.1 and the failures are synchronized to the reformation processes of this script. Every column has a distinct name, so for example the term for the oldest message (which is the first message from left to right in the dataset) is term19, for the time it is time19 etc. The script ends when

the user-set number of term-changes have been iterated and changed, therefore the new data samples have been formed which in this case are 1000 and a new dataset-csv file is exported that is ready to be used by multiple algorithms.

Summary

In summary, a leader in Raft holds its position until a failure occurs that renders the node unreachable to some other node, in which case an election shall commence that will lead to a new leader. The elections are decided by the votes of the node members and the leader is drawn randomly, without any optimization as to which one would be the most efficient choice. However, this could lead to another election if the new leader faces the same problems that the old leader did and to decern which leader would be the best option, the different failures must be distinct. As a result, in this thesis the cluster failures analyzed are link failures and node failures, with link failures being the ones that prevent the transmission of data between two cluster nodes because of a failure in the link connecting them and the node failures being the ones that have to do with the operation of the computer-node (crashes, physical failures etc.). Conclusively, this thesis aims to provide the tools to perform failure prediction based on the messages exchanged between the cluster nodes so that a more optimized leader election can be achieved.

Chapter 3: Machine Learning in Failure Prediction

Introduction

Machine learning is a very quickly evolving field in computer science and the promise is that it can help humanity reach great levels of optimization and accurate prediction in multiple areas of life. Machine learning is already capable of detecting useful patterns in data that humans could not have or would need very long hours of doing so and an example of that is machine learning in tax audits which can accurately and consistently flag tax evading individuals. Deep learning which is another approach in the Artificial Intelligence (AI) field, enables the capability of translating audio to text and vice versa, detecting and identifying objects in photos or videos such as fruit detection in farmer crops or human-animal detection in self-driven cars and many more applications are now possible because of machine learning and deep learning or their encompassing umbrella: AI. That is why this technology is being used in this area, because the prediction of cluster failures and the possible automation of the response to such failures could be detrimental to the stability of systems that more and more encompass people`s everyday lives. These predictions could be made by humans, but the speed and consistency are thought of being incomparable to a machine, let alone a machine learning application which learns by its own mistakes and can improve.

The previous chapters were a theoretical explanation of the subjects and the tools that will be tackled in the experiments that will be discussed in this chapter below, as multiple machine learning algorithms will be trained on the dataset that the scripts from section 2.3 developed and formed, and the results of those experiments will be presented in the sections that follow. The tools utilized to accommodate the failure predictions are machine learning algorithms that train on hundreds or thousands of sets of data-samples that are exchanged between the nodes in cases of failures and in turn leader elections. The machine learning algorithms used for the predictions are decision trees, random forest, support vector machines, adaptive boosting, extreme gradient boosting. These algorithms were all trained on the same datasets so that the results could be comparable to each other, and the best performing algorithms could stand out.

One last note, every single experimentation in the following sections will be of clusters simulated in Mininet with 3 nodes in the clusters, because Raft can only work with an odd

number of nodes, thus the other acceptable configurations would be with five, seven, nine etc.

3.1 Machine Learning

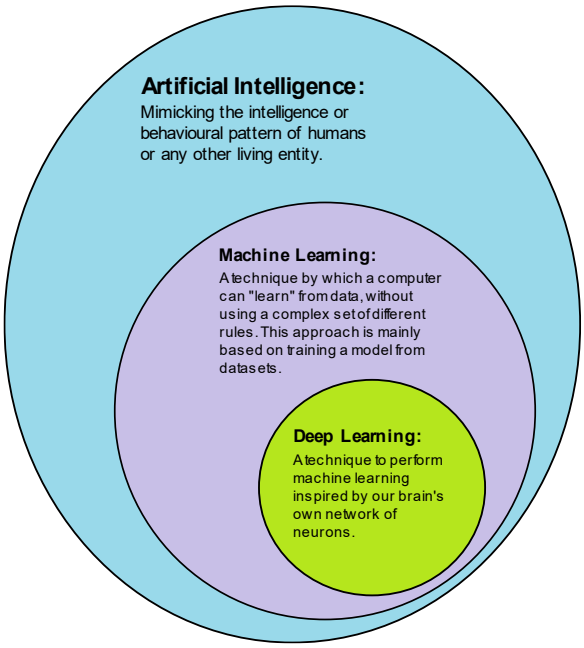

Figure 3.1: How deep learning is a subset of machine learning and how machine learning is a subset of artificial intelligence (AI).

3.1.1 Supervised and Unsupervised Learning

The way machine learning algorithms learn is not the same for every single one of them since some have a supervised learning approach where others have an unsupervised learning approach. The difference between the two is that in the first category the developers have to harvest data, label it, and then feed it to the algorithm so that it can train and provide accurate predictions, however in the second category labeling is not required and instead the algorithm is trying to cluster the data by itself. In more detail, labeling data is the act of giving the dataset the correct predicted value so that the algorithm can train on the given input (one or multiple columns) and attempt to form the correct given output. For example, an interesting way to understand how the training process works is by considering an unknown equation that can be formed if given inputs it exports the desired outputs. The equation would need many different data points so that it can model data more accurately, especially if it is not linear which most machine learning applications are not. In this thesis the algorithms used are supervised learning algorithms, since as it has been explained in section 2.3 the data have the correct failure types that have occurred in the training process

so that the algorithm can learn to accurately predict them. Unsupervised learning was not thought of being particularly useful in this thesis as many of the data samples are quite similar and it was thought that the human label could prove very useful for the data distinction.

3.1.2 Classification and Regression

Another important distinction when using machine learning algorithms is whether the algorithm is a classification, or a regression algorithm and their differences will be explained below. A classification algorithm is used when the predicted variables are in distinct classes, whereas a regression algorithm is used when the predicted variables are continuous values. For example, if the predicted variable is a true or false statement or there are multiple states, but it is required that only one of them be the output then a classification algorithm is used. In contrast to that, if the predicted variable is a continuous number such as money or temperature then the algorithm shall be a regression one. As a result of the above, in this thesis the algorithms used will be classification algorithms since the predicted variable is the "failure type" and that can only be a specific value and not a number in between (a combination of failures is obviously not accepted). In most of the machine learning problems, the predicted value is usually referred to as a "target" value or the "y" from the function denotation in mathematics and in this thesis this variable is the failure type of each sample that was caused artificially in Mininet. The algorithms can not understand from words and phrases what link has gone offline or what node has crashed therefore these values had to be one-hot-encoded which means that every type of failure was translated to an integer starting from 0. That is the way that this field appears in the dataset and that is the appropriate way that target values in classification problems appear in datasets.

3.2 Decision Tree in Failure Prediction

3.2.1 Theoretical Overview

The reason decision trees are very useful in machine learning is because they visualize the results of the training process and can export it as an inverted tree (root is up and leaves are down) so that the developers can understand the decision-making process. Every node of the tree has an expression that attempts to divide the data into groups, to the left of that node it is a positive correlation to that expression whereas to the right of that node it is a negative correlation. Every node`s expression is called an attribute of the decision tree and the leader

node on the top of the tree is called the root node, where every node that does not branch into other child-nodes is called a leaf node with every other node being a decision node. All decisions start from the root node and by evaluating every sample`s correlation to the attribute of the node, the prediction can be obtained by following the directions of the tree and finally to the leaf node which contains the predicted class.

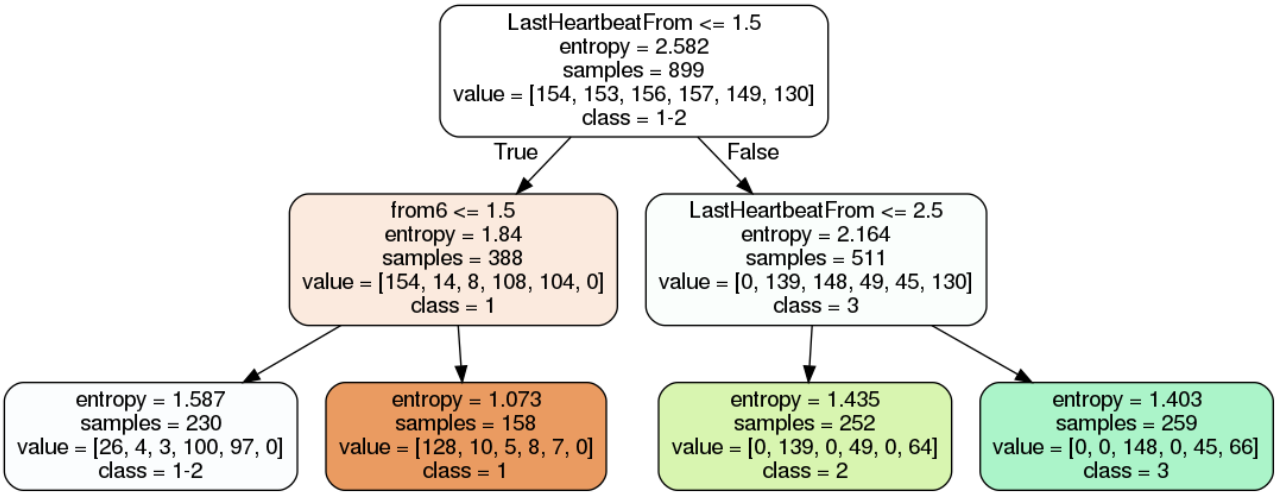

Figure 3.2: A decision tree produced with the dataset of this thesis (even though this tree does not perform well, this is just for visualization). The attribute of every non-leaf node can be seen at the top of each block and the leaf nodes contain the more probable class for that tree path.

The way decision trees work is by using algorithms to decide whether a node shall be split into children-nodes, or it shall remain a leaf node and the way it is decided is by evaluating the node`s purity after a split. Purity is a measure of how the samples in the node are divided in between the predicted classes, and if a split of a parent node leads the children- nodes to have a more "pure" collection of samples in its path then the split is deemed valuable and it`s purity increases, but in decision trees the way this is calculated is by either evaluating impurity or the Gini-impurity[10] or Entropy-Information Gain. Entropy is the measure of disorder or impurity in the samples and if a node of a tree has a somewhat even split of sample classes, then the sample is not pure, since there is no detectable path to a final class prediction and the entropy is increased. If, however, the classes are almost distinct from each other then the entropy is small because there is little disorder in the samples and if one were to pick a datapoint in random it would be more probable for them to pick a specific class. Information gain is equal to 1 – Entropy making it therefore the opposite of entropy, since when entropy is high Information Gain gets smaller and vice versa. For example, if a dataset is targeted at predicting the gender of an input sample and there is information about the

[10] https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity

subject's race in the dataset, then it is obvious that that attribute would not help at all in the decision-making process since race would not determine the subject's gender in a broad and inclusive dataset. Therefore, with the previous example the purity would not increase, or the impurity would increase thus the split would not be a valuable split for the tree to perform and it would be better for that node to be left as it were. It is important to note that the algorithms that decide on splitting do not have to make the best split every time and could be randomized, however there is a parameter which can demand that all of the attributes are considered every time so that the best split can be found. Even though the algorithm sets defaults in many of its parameters the split gain can be overridden by the developer so that the splitting process be made more strict or loose. Additionally, one other area where the tree can be improved upon is the pruning of the tree once it has been fully formed. Pruning as in real-life trees is the act of cutting of branches of the decision tree by evaluating their impurity, as they could be leading the tree to overfitting and not promoting it to grow in a healthier way. In the experimental sections of this thesis, it will be obvious that even though pruning does not enhance model performance, it clears up the trees in an important way making it much easier to inspect the trees. Finally, the mathematics side of the algorithm is out of the scope of this thesis thus for more on the Gini-impurity and the algorithms concerning splitting the reader can be directed here [5].

3.2.2 Decision Tree Training Results

For the training of the decision tree, some hyperparameters have been altered after some experimentation so that the more optimal results could be obtained. Firstly, the *max_depth* of the tree has been set to 6 so that the tree will not overfit to the training data and grow branches that contain needless attribute splits. This number was reached after many tree depths were iterated until there was a plateau in the graphed results as can be seen in figure 3.3 below, even though depths of 9 or 10 show somewhat of an improvement it was considered that the very small increase in testing-validation performance would not be a good tradeoff to potential overfitting issues. To add to that, the *splitter* was set as the "best" as to indicate that the best split will be performed for every node in the tree and the *test_size* split has been set to 10% or 10% of the data is the testing samples and 90% is the training samples as it was the one that produced the best validation model accuracy. Lastly, for the

score accuracy the scikit-learn function *accuracy_score* [11] is being used, as it will in every other training experiment except specified otherwise.

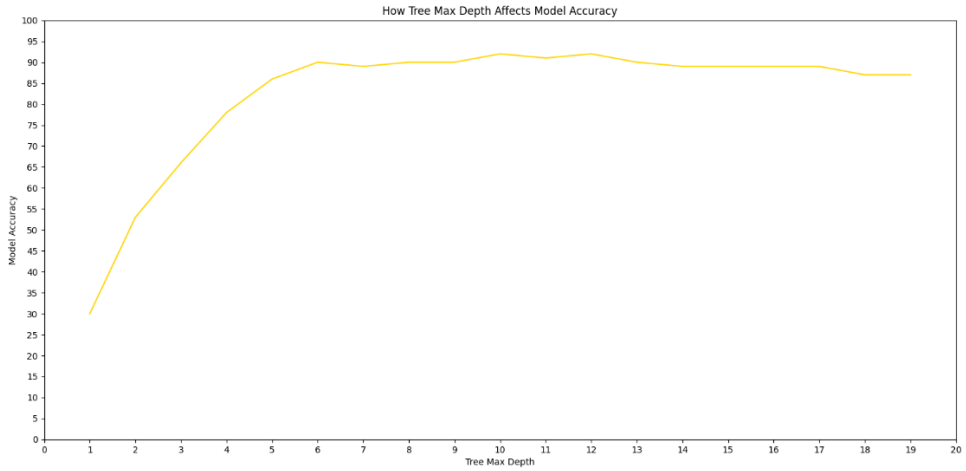

Figure 3.3: A figure produced with matplotlib that shows how the max tree depth affects the testing/validation model accuracy, at some point after a depth of 6 the model accuracy starts to plateau therefore a further increase is not deemed useful.

Finally, another two parameters have been set as to obtain the best results from training with the first one being the *ccp_alpha* (cost complexity pruning alpha) parameter which is the parameter used in order to set the threshold for the branches that have to be cut off if they are not deemed necessary based on the impurity decrease evaluations. The *ccp_alpha* parameter is quite useful for the tree to be simpler and more concise as well as for combating overfitting issues, nonetheless it shall not be set arbitrarily since it can cause a significant loss of performance if set too high (many useful branches will be clipped) as seen in figure 3.4 below. In spite of not losing or gaining any performance by setting the *ccp_alpha* parameter at a very small value, (based on figure 3.4 the minimum value is 0.01) one would wonder why it is being used in the first place and the answer can be observed by the visualization of the trees produced by the decision tree algorithm in figures 3.6 and 3.7. The latter is the *min_impurity_decrease* parameter which is the minimum threshold for the impurity decrease of a node when it is split and if it is lower than the threshold the split is not performed, and some experimentation can be seen in figure 3.5 below. Lastly, it shall be noted that Entropy is being used instead of Gini-impurity for the training results of this section as it was seen to show significantly better results. The algorithm used was in python

[11] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

from the package scikit-learn[12] seen in [6] where exemplary code and instructions to use the decision tree classifier[13] were given for anyone to use for free.

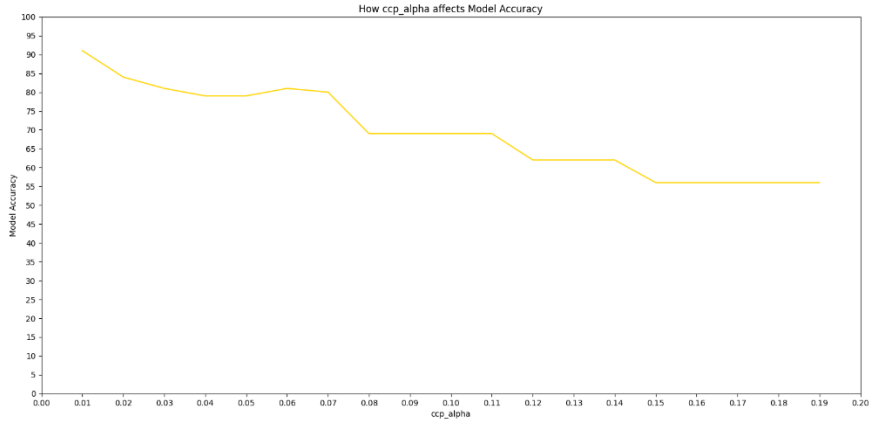

Figure 3.4: The model validation accuracy based on the increasing ccp_alpha parameter. The model reaches peak performance right at the very beginning of this parameter, which is expected since the more branches the tree has the more cases it can cover, whereas the greater the ccp_alpha the more branches are being cut off.

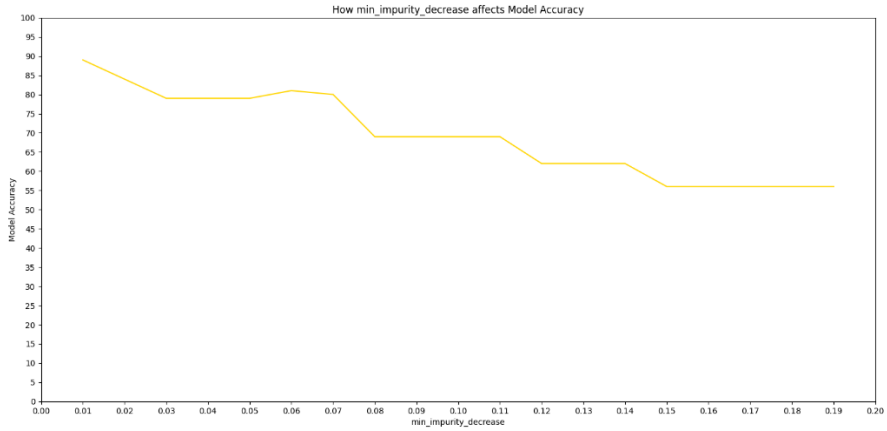

Figure 3.5: The plot of how the increasing min_impurity_decrease parameter affects the model validation accuracy if max depth is 6 and ccp_alpha is 0.01. It is almost identical to figure 3.4 but has some sharper lines, and this can be expected as both parameters work with the impurity decrease of a given split or branch.

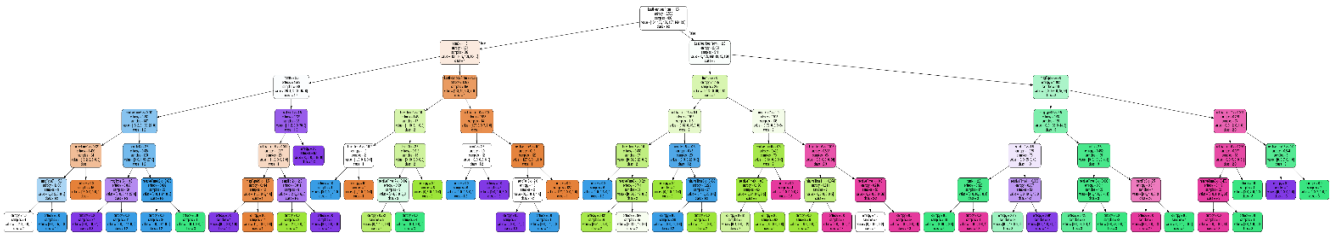

Figure 3.6: A visualization of the decision tree produced when ccp_alpha is set to zero and the max tree depth is set to 6.

[12] https://scikit-learn.org/stable/index.html
[13] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

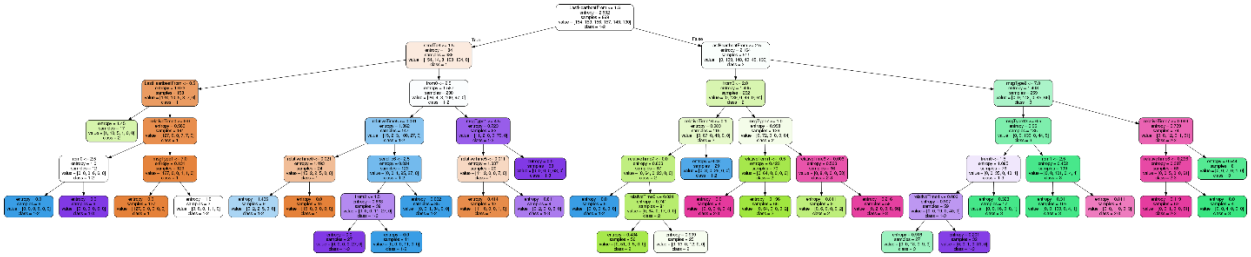

Figure 3.7: A visualization of the decision tree produced when ccp_alpha is set to 0.01 and the max tree depth is set to 6.

The model with a ccp_alpha of 0.01 and a min_impurity_decrease of 0.01 with a max_depth of 6 gave the best validation results of 90% of classification accuracy with a 93% training accuracy. Although the aforementioned parameters helped in the visualization of the tree (one or the other they did not have to be used together as they produced nearly identical results) they did not help with the performance of the tree in any significant way, and that may have to do with the fact that the tree probably did not overfit to the training data since the max_depth was not unlimited.

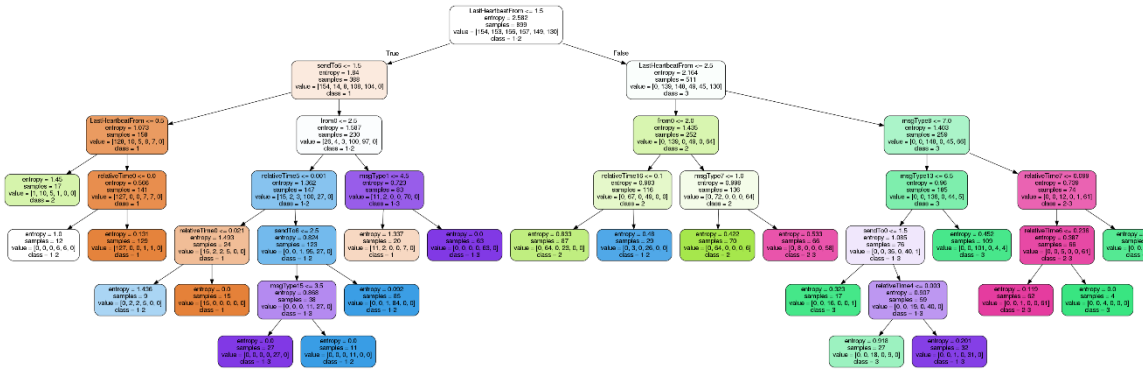

Figure 3.8: This is a visualization of a decision tree with a min_impurity_decrease of 0.02 and a ccp_alpha of 0.01. It is very clear that the more these parameters are increased the simpler the trees become, with a cost in performance however as this iteration had an 84% classification score instead of a 90% that the previous model had.

3.3 Random Forest in Failure Prediction

3.3.1 Theoretical Overview

The closest algorithm that could provide a better accuracy than simple decision trees is a random forest algorithm, since it is built for being more robust. The way the random forest algorithm works is by training multiple decision trees by using random sets of the features of the dataset (not necessarily all of them) and by giving the input samples to all of them and then getting predictions by every-one of them. The trees, therefore, vote what the final predicted class is which is considered much more robust, since at the end of the prediction process there are multiple votes in the pool and if only some of them have overfitting problems or they have low accuracy then the other ones can make up for that.

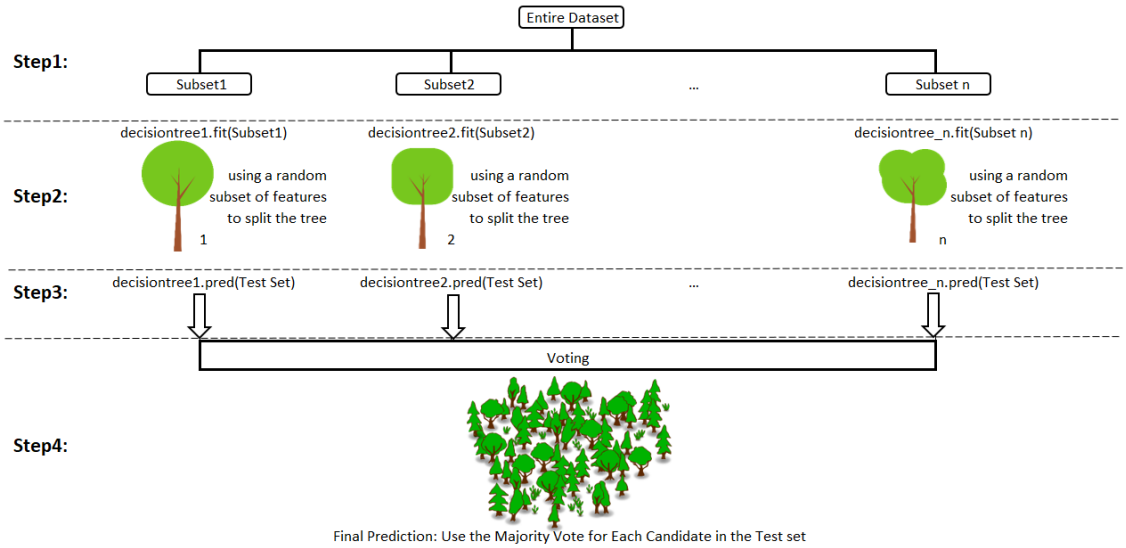

Figure 3.9: A random forest algorithm training visualization. More decision trees are considered more robust than just a single one. Graph taken from https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0

Unlike simple decision trees, in random forest the splitting is random, and it is not concerned to find the best split, which in turn enhances the randomness of every tree so that the model has very differently built trees that will hopefully predict the same class. The number of the trees used in the forest can be set by the user, however the more the trees the more the computational complexity the algorithm requires, thus this parameter shall not be abused as the algorithm will be rendered impossible to run in real-time. Consequently, one could point out that random forest takes advantage of multiple other classifiers and brings them together for its own prediction and thus this method is called an "ensemble" method and it is possible to utilize multiple classifiers and not just decision trees. Furthermore, the method of training every model in parallel is called "bagging" in machine learning terms and it entails that every classifier in the forest is trained on a random subset of the data. Unlike random forest there are classifiers that utilize ensemble methods but without the use of bagging but with the utilization of boosting which will be described in section 3.5.

3.3.2 Random Forest Training Results

As the hyperparameters are concerned, the *n_estimators* parameter is the number of trees in the forest, and it was set at 200 trees after some experimentation on how it affects model performance (see figure 3.10), but as far as the *min_impurity_decrease* and ccp_alpha are concerned their values were kept the same as in the decision tree section in 3.2.2 since the main classifier is the decision tree and both of them were used since it appeared that a consistent increase of 1 or 2 percentage points of validation accuracy were gained. The

max_depth was not set and was left to default which is “None” or unlimited and the *test_size* was left to 10% and the criterion for splits was the “entropy” method. For the training part of this section the scikit-learn package was used and more specifically the random forest classifier[14], and some of the code that is presented proved useful to train the dataset.

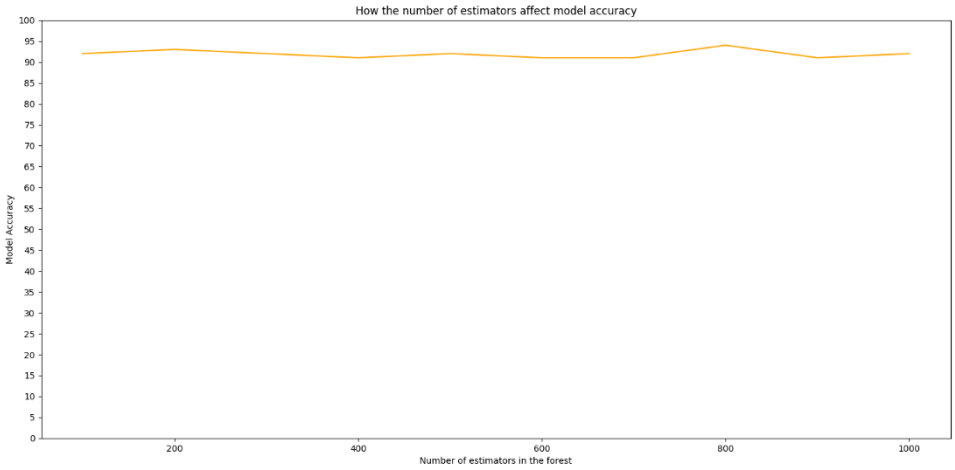

Figure 3.10: The plot of how the number of estimators affect the model performance. The plot shows that the validation accuracy is not affected too much from the number of estimators and this could be expected since the original decision tree performance was very good.

As for the final model performance, the random forest had a validation performance of 91% up against a training performance of 97% which shows that there is still some form of model overfitting because of the difference in the two accuracies.

3.4 Support Vector Machines in Failure Prediction

3.4.1 Theoretical Overview

The Support Vector Machines algorithm or SVM is essentially plotting every single point of the sample or every feature-column field of the sample to an N^{th}-dimensional space when the features are N, so in this case in the 20^{th} since the features are 20. The classification part and the training part of the algorithm is trying to form a hyperplane that can segregate the data in groups according to the labels the algorithm has been given. The optimal solution of the hyperplane is the one that has the maximum distance between the data points and the hyperplane. As a result of that, it is obvious that there is no visualization for the training results of SVM`s for datasets with more than 3 features since there is no visualization of anything beyond the 3D space. Finally, the SVM classifier was taken from the scikit-learn package the link of which can be found here[15] where the mathematics side of the classifier

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

is described in detail as well as code examples are presented and the hyperparameters are explained.

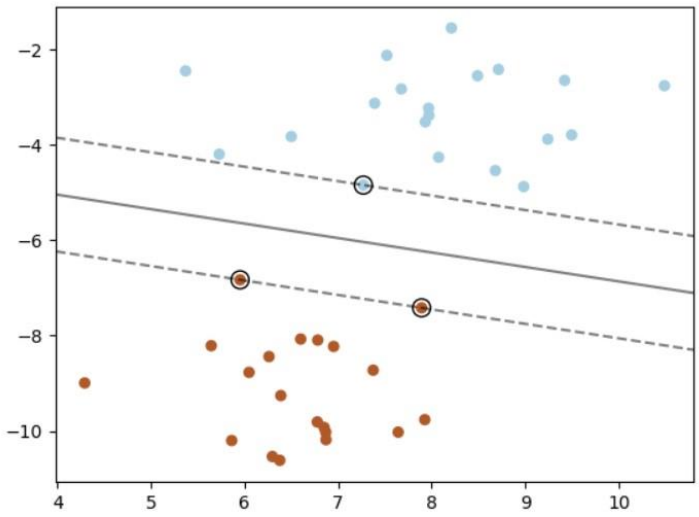

Figure 3.11: The attempt to find a hyperplane for a dataset with 2 features so that the samples can be divided in different groups and the prediction process can be accurate. Taken from https://scikit-learn.org/stable/modules/svm.html

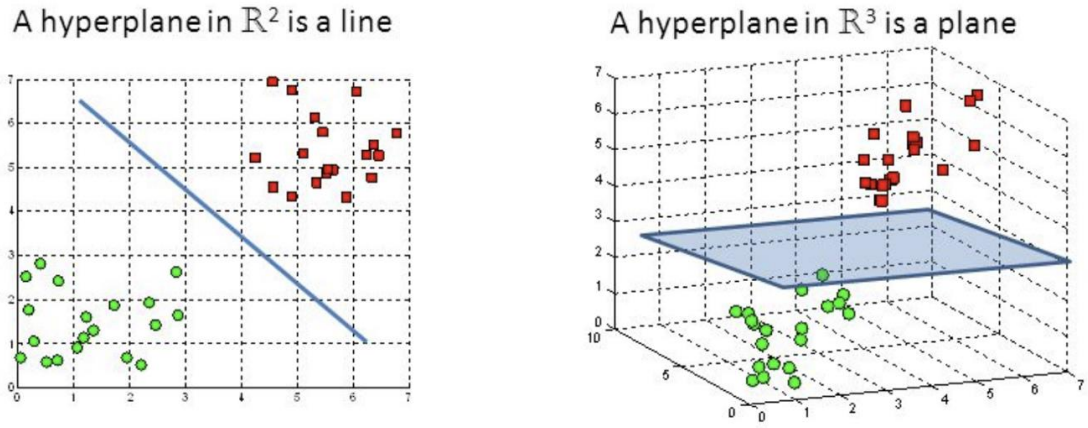

Figure 3.12: A visualization of a 2D line and a 3D plane that segregate the data into their groups so that accurate predictions can be possible. Graph taken from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

3.4.2 Support Vector Machines Training Results

As for the training results for the SVM algorithm there were 2 runs, the first one with the SVC with a linear kernel and a parameter of regularization C = 1.5 (after some experimentation it was the optimal choice) and that model gave a 98% training accuracy with an 89% validation accuracy. The latter run was with the model is with the LinearSVC model with parameters: penalty = 'l2' and a loss function of "squared_hinge" with a dual of False and a C of 0.2. This iteration gave an 84% validation accuracy and a 92% training

accuracy. Therefore, the first run was the better model of the two, however SVM did not show any improvement over the decision trees or the random forest.

3.5 Adaptive Boosting in Failure Prediction

3.5.1 Theoretical Overview

Adaptive boosting is another ensemble algorithm that uses decision trees as its main classifier, even though that combination looks a lot like the random forest algorithm unlike the latter this algorithm as its name suggests uses boosting instead of bagging which means that the way the model trains is much different than the parallelized way that random forest uses. In boosting the algorithm takes the previous classifier`s mistakes and uses them for its own training and this happens sequentially for every decision tree in the algorithm, therefore at the end of the training process the algorithm has tried to fix many of the errors and mistakes of the original trees. More specifically, after the first tree has been trained the wrongly predicted data points get a weight greater than the others (the original weights are 1/N where N is the number of samples) and the greater the tree weight the less decision power that tree gets in the final voting process, hence the lower the weight or the more correct predictions a tree makes the more leverage it gets towards the final prediction. Obviously, this algorithm can not run in parallel thus it will take longer if the number of trees is big, and it might not be possible to run in a real-time scenario.

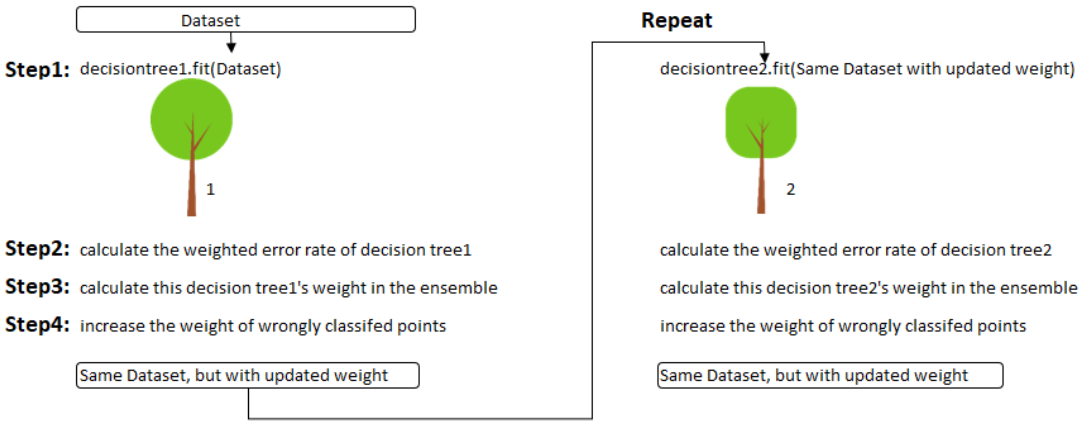

Figure 3.13: A visualization of the training process of adaptive boosting. The miscalculations of the first trained tree come into play in the training of the next tree and so on. Graph taken from https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725

3.5.2 Adaptive Boosting Training Results

The algorithm can be obtained by the scikit-learn website and in this thesis the adaptive boosting classifier[16] is used. As far as the parameters are concerned, the *max_depth* was set to 6 and the *ccp_alpha* to 0.1 with *n_estimators* being 200 and the *learning_rate* left to the default value of 1 after experimentation showed little to no gains if it changed (see figure 3.14) and lastly the *test_size* was kept at 10% of the dataset. Most of the parameters have already been introduced, the *learning_rate* however has not, so what this parameter does is it affects the magnitude of the weights of each boosting iteration. The greater the learning rate the greater the contribution of the trees hence this parameter shall be used with caution as the weaker trees could have more decisive power if its value is increased too much from its default value of 1.

The model results were the best until this point of the experimentation with a 92-93% classification accuracy for validation testing and a 95-96% accuracy for training. This is to be expected as the ensemble methods are much stronger than the simpler models, however since the original "weak" predictors were not as weak, it is not a great performance uplift.

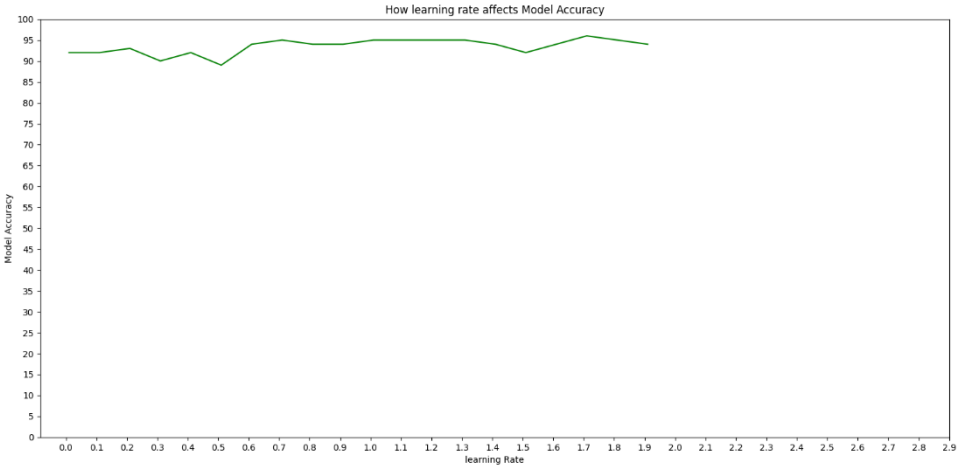

Figure 3.14: The plot of the learning rate and how it affects the model performance. With the learning rate having a default value of 1 the model shows little to no performance gains when the parameter is around that value.

3.6 Extreme Gradient Boosting in Failure Prediction

3.6.1 Theoretical Overview

Extreme Gradient Boosting or Extreme Boosting is quickly becoming very popular and very competitive in the machine learning competitions online because as its name suggests it pushes to the extreme what is possible with boosting algorithms. Even though the main idea

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

is the same as with adaptive boosting algorithms which have not been explored in this thesis, extreme boosting uses a gradient as its loss function instead of the Gini impurity or entropy as simple decision trees do and as the author of the algorithm has stated: "… xgboost used a more regularized model formalization to control over-fitting, which gives it better performance."[17] when prompted to compare the two. Extreme boosting relies heavily on mathematics and that is why it is performing well in competitions as well as running faster than other algorithms while at the same time utilizing less resources. Nonetheless, more on how extreme boosting works (or xgboost as it is often called) can be seen here [7].

3.6.2 Extreme Boosting Training Results

Finally, as for xgboost, the scikit-learn API was used along with the xgboost package[18] taken from PyPi[19] which contains the steps on how to install and redirects the users to the documentation[20]. As for the parameters, the number of estimators has not really impacted the performance of the models too much and that is because the numbers used were above at least 100 estimators which is usually good enough, besides that in this implementation 400 estimators were set after some experiments. The max depth was set to 6 as every other tree-related algorithm and the eta or the learning rate version of xgboost was set to 0.3 again after some experiments were run and their results can be seen in figures 3.15 and 3.16. The *objective* parameter was set to "multi:softprob" as no difference was shown whence it changed. Lastly, the actual model performance results were the best out of all the algorithms compared in this thesis, and unlike some other algorithms xgboost did not require a lot of configurations before it performed in the top echelon of the competition. More specifically, the model had a 100% training accuracy which gives red flags as to overfitting, however the training accuracy reached 96%. Although the tree depth does not increase performance by a lot, as seen in figure 3.15 there is a maximum for tree depth of 9 and by making this change the model can reach a maximum of 97% of testing/validation accuracy. Conclusively, the algorithm was the most stable and consistent out of all the algorithms as well as the fastest of the ensemble algorithms, maybe even all.

[17] https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting/answer/Tianqi-Chen-1?srid=8Ze
[18] https://pypi.org/project/xgboost/
[19] https://pypi.org
[20] https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier

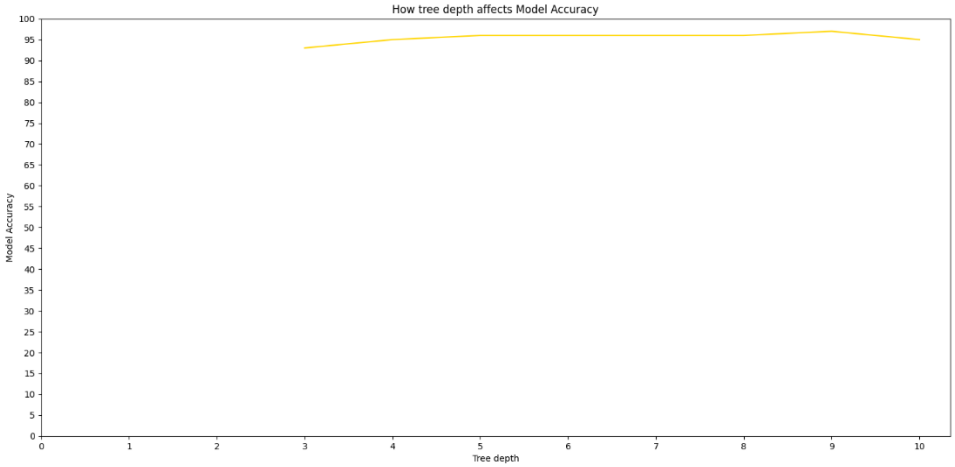

Figure 3.15: The plot of the model and the effects of the tree depth increasing. As expected, the tree depth does not influence the model too much as long as it is in the range of 4 to 10 levels. However, there is a maximum in the performance in the tree depth = 9.

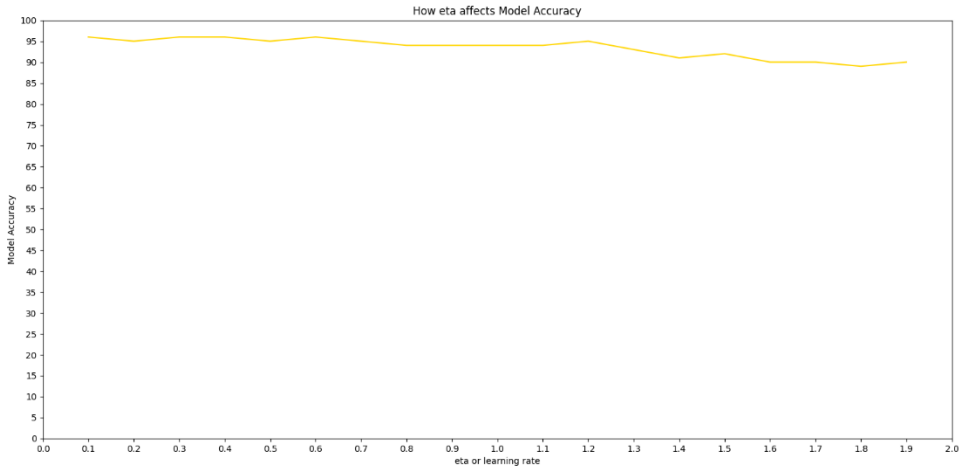

Figure 3.16:The learning rate or the version of learning rate for xgboost "eta" and its effects on model performance. It appears as if the model performs very well for anything below 0.6 for eta with a maximum at around 0.3.

Conclusions

In this thesis the subjects of computer clusters and orchestration systems was tackled, and machine learning tools were developed and used in order to improve the Raft algorithm which handles the etcd elections in a cluster environment. After using a Mininet simulation with the raft-example implementation of etcd and by using custom scripts that enable artificially caused failures so that the communication messages between the cluster-nodes could be extracted, machine learning algorithms were used upon these datapoints with good performance. All of the models reached very good levels of performance but the standout algorithm in both terms of configuration and classification accuracy was the Extreme Gradient Boosting algorithm or xgboost as it is often referred as. This model outperformed all the other models and was by far the most consistent in its performance and it is estimated that it is a very good candidate for any future work on this subject, even in real-time performance evaluations. All the simulations used three nodes in the clusters, however with some configurations the work of this thesis could be expanded for five, seven, nine and so on, with another improvement being the utilization of the full version of etcd and not of the raft-example version. Conclusively, future work could be done based on this thesis so that not only more nodes are in the training dataset and thus the machine learning experimentation, but so that the benefits of such predictions could improve and influence the Raft elections in a real Kubernetes environment and its performance benefits could be measured in a more realistic scenario.

Bibliography

[1] R. Long and M. Różacki, "https://cloud.google.com," Google.com, 23 6 2020. [Online]. Available: https://cloud.google.com/blog/products/containers-kubernetes/google-kubernetes-engine-clusters-can-have-up-to-15000-nodes. [Accessed 15 9 2021].

[2] "https://www.ibm.com/cloud/learn/etcd," https://www.ibm.com, 18 12 2019. [Online]. Available: https://www.ibm.com/cloud/learn/etcd. [Accessed 15 9 2021].

[3] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," Stanford University, California, 2014.

[4] R. VAN RENESSE and D. ALTINBUKEN, "Paxos Made Moderately Complex," 17 2 2015. [Online]. Available: https://doi.org/10.1145/2673577. [Accessed 15 9 2021].

[5] N. S. Chauhan, "kdnuggets," https://www.kdnuggets.com, 14 1 2020. [Online]. Available: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html. [Accessed 15 9 2021].

[6] F. P. e. al, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[7] . T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," 13 8 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939785. [Accessed 15 9 2021].