School of Computer Science and Informatics

CM3203 – Individual Report

Comparative Analysis of Tree-Based Machine Learning Models and Neural Networks for Malware Prediction

Author: Kriti Shewaramani - 1845909

Project Supervision: Yuhua Li

Project Moderator: Sandy Gould

A Final Year Project Submitted for the Degree of BSc. Computer Science with Security and Forensics with Placement Year

2022

ABSTRACT

In the age of technology, malware attacks are occurring every day around the world. These attacks involve malicious software that can lock up essential files, spam you with ads, or redirect you to malicious websites, which can result in anything from data theft to the destruction of entire systems or devices. Cybercriminals use different types of malware like trojans, ransomware, spyware, or worms to infect individuals or organisations. Machine learning is a credible technology in today’s day and age. The concepts of machine learning can be applied to the process of malware detection in order to efficiently detect and prevent malware activities. The project aims to apply machine learning to predict a computer’s probability of getting infected by various families of malware based on different properties of that machine using three types of machine learning models: namely, LightGBM, XGBoost, and Neural Network. These models are trained on a dataset published by Microsoft of Kaggle with over 80 features from the reports from Windows Defender. Before implementing the models, data pre-processing, feature engineering and exploratory data analysis were carried out. Once successfully implemented, these models then find similarities and patterns between the data to perform Classification. Python programming language and Jupyter Notebook were used during the entire duration of the project.

ACKNOWLEDGEMENTS

Firstly, I want to thank my supervisor Dr Yuhua Li, who has been exceptionally helpful throughout the semester. He has constantly helped me understand and achieve my short-term and long-term goals in this dissertation. With his support, I was able to balance my work and also apply to and secure a job in Amazon Web Services in London.

I would like to thank my family for inspiring me to perform better every day. They have been a constant source of encouragement in these challenging times, and I would have been able to successfully complete this project and the last four years without their support.

I would last want to thank my friends who helped me focus on my work and created an environment that was a home away from my home. I will forever be grateful for all the memories I made and those who have been by my side.

Table of Contents

Index of Figures and Tables

1. Introduction

In the 21st century, the internet has become a necessity for individuals and organisations. There is an exponential growth in cyberattacks, especially during the COVID-19 pandemic, cyberattacks have increased by 600% (Helen, 2021). Hackers are designing malicious software to install into networks or the victim's machine. This malicious software is called malware. It is an intrusive software designed to damage and destroy computers and systems. Malware can crack weak passwords, bore into systems, and spread through networks.

Everyday windows users are a target of a huge number of malware designed by attackers looking to find vulnerabilities. This is where Malware detection comes into place. Malware detection is the process of scanning the computer and files to detect malware. It effectively mitigates a possible security breach because it involves multiple tools and approaches. There are multiple approaches to detecting malware, like sandboxing, where we detect malware by testing a code in an isolated environment; heuristic analysis, where a baseline for standard activity is defined for a system and any deviations in behaviour are observed. There is also Endpoint Detection and Response, EDR, which monitors and logs events from the endpoint, this data can be used to analyse the behaviour after the computer is infected by malware. Lastly, machine learning malware detection can be used to train systems to differentiate between malware and benign files and predict if the computer has malware or not. Using machine learning models that learn and improve constantly would add a higher level of security in networks and systems as it can keep pace with malware development.

1.1. Research Questions

This project aims to explore the following research questions:

Research Question 1: Can machine learning algorithms be used to predict if a windows machine has malware?

Research Question 2: Which machine learning binary classification model performs best in predicting if a windows machine has malware?

These questions will be explored by delving into machine learning for malware prediction by implementing different Machine Learning classification models and evaluating metrics to identify the best technique to solve this problem. To train these models, a telemetry dataset containing these properties of windows computers and the machine infections will be used. This dataset was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender.

1.2. Aims and Scope

The primary aim of this project is to implement a machine learning model that gives the best results in predicting a computer's probability of having malware using the properties of a Windows computer.

This aim will be achieved by investigating and evaluating the performance of different classification algorithms to identify the best machine learning model. This project's scope expands to examining which Machine learning classification model is efficient in terms of time, accuracy, precision, and other metrics. The novel approach here is implementing three different algorithms in order to compare tree-based models to artificial neural networks. The algorithms discussed in detail further are LightGBM, XGBoost and Neural Network. The models were chosen based on the literature review and the popularity of different machine learning forums. The hyperparameters will be tuned for each of these algorithms to achieve the highest possible accuracy. This is discussed in detail in the Implementation (Section 4) and results (Section 6)

1.3. Intended Audience

The intended audience for this project is researchers, students, analysts, or individuals interested in understanding more about the domain of the application of machine learning algorithms for malware prediction.

1.4. Report Structure

The rest of this report is divided into multiple sections, as explained below:

Section 2 throws light on the background research undertaken to understand the fundamentals of the problem. In this section, an extensive literature review is conducted, exploring the history of cyber security, machine learning and prior research that exists within the domain of cyber security in Machine learning. Complementing this is a systematic literature review, and an in-depth critical analysis of specific literature has also been presented. **Section 3** discusses the approach and provides an explanation of the steps taken to complete this project, giving a clear description of the problem and the machine learning algorithms that have been implemented along with the methodology used to solve this problem. **Section 4** is Implementation; this section takes a deeper look into how machine learning models can provide the best solution to the problem and highlights key decisions that were taken in the Implementation of the code. **Section 5** focuses on testing and depicts the results of the implementation stage using a well-devised testing plan that has been used to evaluate the project along with other testing techniques. The results have been critically analysed and discussed. **Section 6** discusses the results and evaluation to measure the success of the project by critically analysing and evaluating the results of the machine learning models that have been implemented. **Section 7** contains the conclusion and future work, i.e. an in-depth assessment of the research that has been conducted is provided, accompanied by limitations of the project and future work that can be considered to improve the research. **Section 8** finally contains a Reflective piece on knowledge gained, challenges faced, and improved alternatives learned during the completion of this research.

2. Background

2.1. History of Cyberattacks

A cyber-attack is a deliberate assault launched by cybercriminals against individuals or information systems (NCSC, 2019), Cybercrime has been increasing ever since people have been trying to benefit from vulnerable networks and systems. An attack carried out by a skilled individual may be targeted and contain multiple levels.

Cyber-attacks are evolving continuously by becoming more and more sophisticated. (Climer, 2018) The first denial of service (DoS) attack in history was in 1988, when Robert Morris created a computer worm and slowed the early internet down. This worm was not written to damage but to draw attention to security issues like weak passwords. However, the code managed to duplicate the worm exponentially and caused not only damages worth $10,000,000 but also a partition of the internet for several days.

In the following year, 1989, the first Ransomware attack surfaced. AIDS Trojan was created by Joseph Popp to extort people for money. This was dispersed through a floppy disk and was easily removable due to poor design – in the sense that it was only scrambling the names of the files rather than the contents, which made the files usable.

The next year, 1990, marked a significant point in the history of cyberattacks. The Computer Misuse Act was passed in the United Kingdom (Townsend, 2019) – which stated that unauthorised attempts made to access computer systems are illegal. This act is still in practice with some modern additions and amendments made through these years.

In the current time, malware circulated has exponentially increased with the use of exploit kits and automated SQL injections. The next sections discussed the history of Malware attacks in further detail.

2.2. Introduction to Malware attacks

Malware is an amalgamation of the term malicious software. This is an application created by an individual or a group with malicious intent. Malware can include spyware, ransomware, viruses, and worms. Malware breaches a network through a vulnerability, usually with hidden goals like blocking access to key network components, installing destructive software, stealthily obtaining information by transmitting data from the hard drive, disrupting certain components, and rendering the system inoperable (Mary Landesman, 2021).

In the early years, spreading malware was done manually by human hands; this meant carrying floppy disks from one computer to another. As the internet matured, the idea of hackers getting malware onto as many computers came into practice. This extends back to the 1970s when a self-replicating program called the Creeper

Worm gained access via the ARPANET and copied itself to remote systems. From then until 2000, there were a few primitive malware attacks like Wabbit, another self-replicating program that reduced system performance. Brain Boot Sector Virus was the first virus to affect MS-DOS computers. PC-Write was the first trojan that was disguised as a shareware program, and it would erase everything on a computer once installed by the user. However, 1988 was the first time someone was convicted of authoring malware; Robert Morris created the Morris worm that affected most machines on the ARPANET and rendered the network unusable within 24 hours.

From 2000 (Rankin, 2018), malware attacks grew significantly in speed and number. This decade also marked the growth of email worms and malware toolkits. SQL injections were also rising and, at a point, became the number one threat to individuals and organisations. The infamous ILOVEYOU Worm was an email virus that shut down the email servers of over 50 million devices, including huge government bodies like the Pentagon and British Parliament: causing global damages worth 5.5 billion dollars. There was then the SQL Slammer worm which affected 75000 devices in 10 minutes and caused a worldwide slowing down of internet traffic via denial-of-service.

In the next decade, we noticed a rise in state-sponsored malware along with more sophisticated and profitable attacks. Malware attacks have become more targeted and well-funded for malicious groups that continuously work to develop programs that can outsmart anti-malware systems. The beginning of this decade, 2010, marked the release of Stuxnet Worm. An entire team of developers designed this malware to attack Iran's nuclear program and damage not only the hardware but also the software. It has by far been called the most resource-intensive malware to date. We also saw one of the first ransomware programs, called Cryptolocker – this program utilised a trojan that targeted all computers running Microsoft Windows. As of recent, we heard about the WannaCry Ransomware, which was discovered by the National Security Agency and brought 150 countries, including Russia, China, the United Kingdom, and the United States of America, to their knees. This ransomware locked individuals and organisations out of their personal data and kept it so that until a payment was made in case the user refused to make a payment, they would lose all their data. It was especially difficult for industries like medical, financial and telecommunications.

Currently, we are seeing organisations deploy anti-malware solutions that help users avoid most of the damage from malicious software (Microsoft, 2022). Windows Defender is Microsoft's endpoint protection solution. Microsoft Defender Antivirus is preinstalled into Windows 10, Windows 11, Windows Server 2019, Windows Server 2022, and Windows Server 2016. This program helps review the security status of a device and receive alerts in case of an issue. Microsoft endpoint manager can also be used to monitor the Defender Antivirus.

2.3. Introduction to Machine Learning

American pioneer Arthur Lee Samuel first coined the definition of machine learning in 1959. He stated, 'Machine learning is the field of study that enables computers to learn without specifically being programmed.' A more formal definition was written in 1997 by T. Mitchell, stating, 'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.' The basis of Machine Learning is to train a model based on an algorithm most suited to the task at hand. In this project, the task is Classification. To perform Classification, ML supplies an automated, adaptive approach that can extract data provided in the form of training sets using a model and then use this information to classify new data (Sidath Asiri, 2018).

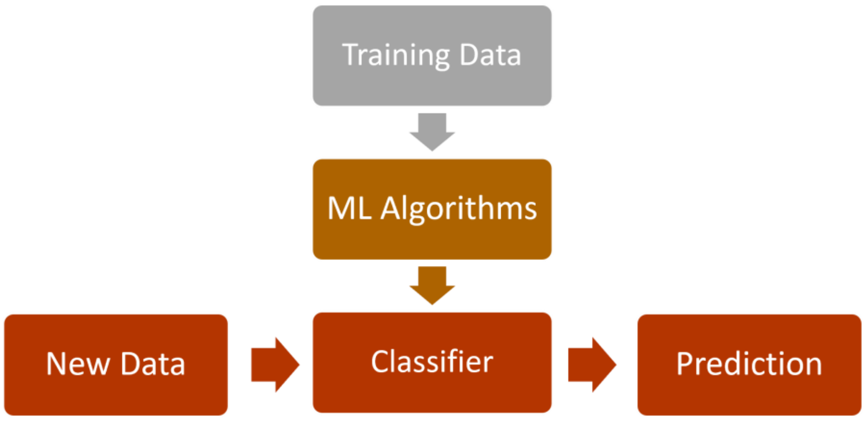


Figure 1 Machine Learning Process

In recent years, the emergence of Machine Learning algorithms has become an essential tool in the field of Cyber Security. With the ability of machine learning to be leveraged to improve malware detection, triage events, recognise breaches and alert organisations of security issues, the cyber security industry can benefit, and its landscape can be altered, benefiting individuals and significant corporations around the world.

2.4. Use of ML for malware detection

Machine learning is an ever-evolving field; I will further discuss how it uses algorithms to process large sets of data. In cybersecurity, machine learning is helping us develop more sophisticated cybersecurity tools to recognise patterns, learn from them and prevent future attacks using threat hunting. There are multiple use cases for machine learning enhancing threat detection. For instance, cutting down false positives – Machine learning-based security programs do not interrupt the flow of traffic on a system as they can make smart decisions and block out malicious software without depending on rulesets. Another use case would be noting inconsistencies in patterns of transmitted data during threat hunting - The algorithm might not recognise the inconsistency as a known threat, but the inconsistency could trigger further investigation. Malware prediction modelling is the key use case under Artificial Intelligence for Cybersecurity (Holmen, 2022) – supervised ML can train machines to detect harmful files, create a model of what the malware looks like and then block that malicious software in the future. This use case certainly has some drawbacks, like not being able to account for malware variants, however, it can constantly learn from newly updated data and revise the model as and when required.

Almost every cybersecurity solution is basically learning from past and current issues and using that knowledge to improve future protection (Fidelis Cybersecurity, 2020). This is where machine learning comes in, in the sense that we are able to learn and operationalise 'patterns' that help us understand what is malicious. Known hashes of malicious programs, websites hosting malware, and email addresses used for phishing and spamming are all part of the knowledge we need for future protection. This knowledge is very large, and identifying, classifying, and operationalising millions of such files a day is difficult. However, to solve this issue, we have Supervised Machine Learning Algorithms that can create models with rules that would be able to classify known malicious programs from non-harmful programs. Supervised Machine learning can also be predictive, which would identify malware that has not been sampled or known beforehand.

The use of supervised machine learning for detecting malware files is based on different aspects of the files (structure of the file, content in the file, use of different characteristics in file formats, behavioural patterns of the file when executed or launched in application) has become common in the industry and has conclusively demonstrated to be rather successful in detecting future threats. There are numerous other applications for Supervised ML, such as identifying if new domain names and certificates closely match known malware activity.

Instead of pattern recognition, we could also have looser rules of recognition, which means that an operator can discover a pattern that is not 'consistent' in time, therefore, hinting to a potential threat, thereby encouraging the operation to investigate a possibly malicious event. To find these patterns that are inconsistent, unsupervised machine learning algorithms can be used to create a standard or basis of the usual network traffic activity for users. This standard can then be used to compare and detect unusual activity that may lead the operator to conduct further investigation to understand if it is truly a malicious threat. The use of unsupervised machine learning algorithms in the current age requires a threat hunting platform in order for an operator to spot anomalies and further hunt and investigate those to detect threats. There may, however, be several anomalies that have no support from a threat hunting platform. This makes them of no value during threat detection and may create a large number of false positives that are of no use to the operators. On the other hand, if an anomaly has evidence and data to back up the investigation, it can be used to advance threat detection and reduce the impact on an individual or an organisation.

2.5. Literature review

In this section of a systematic review, web sources such as Google Scholar, Kaggle, and GitHub have been used to create a literature review. A total of 63,600 results were returned when the query ‘Machine Learning for Malware Detection was searched in the primary web source Google Scholar. The search was further concentrated by additionally specifying machine learning models like ‘LightGBM’, ‘XGBoost’ and ‘Neural Networks. Another query to find similar pieces of literature was 'Machine Learning for Malware Prediction'; this search returned 36,700 results. Eventually, nine suitable pieces of literature were identified in Table 1. The table features the authors, the title of the study, and the dataset used and recognises the Machine Learning models and the outcomes highlighted.

Table 1 Systematic Literature Review

Authors	Title	Dataset	ML Algorithm	Outcomes
Leevy, J.L., Hancock, J., Zuech, R. et al.	Detecting cybersecurity attacks across different network features and learners (Leevy et al., 2021)	CSE-CIC-IDS2018 dataset: 16 million instances	Decision Tree, Random Forest, Naive Bayes, Logistic Regression, Catboost, LightGBM, or XGBoost	Feature selection technique yields performance like, or better than, using all features. Including the ‘Destination_Port’ feature has a significant impact on performance in terms of AUC.
Al-Omari, M., Rawashdeh, M., Qutaishat, F. et al.	An Intelligent Tree-Based Intrusion Detection Model for Cyber Security (Al-Omari et al., 2021)	UNSW-NB 15 dataset from the Cyber Range Lab of the Australian Center for Cyber security with 175,341 records.	Tree-based model (Intrusion detection tree)	The paper presented an intelligent tree-based intrusion model that was able to effectively and efficiently predict/detect cyberattacks.
Rajesh Kumar, Geetha S	Malware classification using XGBoost-Gradient Boosted Decision Tree (Kumar and S, 2020)	EMBER dataset consists of 1.1 million entries with a label for malware, benign, and some parts left as unknown	Classifier using XGBoost: Extreme Gradient Boosting	A model was trained using low computation resources in 1315 seconds with a reduction in the feature set. The hyperparameter tuned model gives improved

				performance for accuracy of 98.5 and on par AUC of .9989.
Evgeny Burnaev, Dmitry Smolyakov	One-Class SVM with Privileged Information and Its Application to Malware Detection (Burnaev and Smolyakov, 2016)	Microsoft Malware Classification Challenge dataset: 0.5TB, consisting of disassembly and bytecode of more than 20K malware samples	Combining Support Vector Machines and learning using privileged information and training SVDD and One-Class SVM	Privileged information can significantly improve anomaly detection accuracy.
Qiangjian Pan, Weiliang Tang, Siyue Yao	The Application of LightGBM in Microsoft Malware Detection (Pan, Tang and Yao, 2020)	Microsoft Malware Prediction dataset on Kaggle: 9 million rows and 83 attributes	Logistic Regression , KNN, Light Gradient Boosting Machine	LightGBM is a gradient boosting framework that uses the tree-based learning algorithm, and it shows the best performance among the three models
McLaughlin, Niall and Martinez del Rincon, Jesus and Kang et al	Deep Android Malware Detection (McLaughlin et al., 2017)	Raw Dalvik bytecode of an Android application	Deep Convolutional Neural Network	The paper presents a novel Android malware detection system using deep neural networks and achieving 69% accuracy using cross-validation.
Taeshik Shon, Yongdae Kim, Cheolwon Lee and Jongsub Moon	A machine learning framework for network anomaly detection using SVM and GA (Shon et al., 2005)	MIT Lincoln Lab Dataset	Genetic Algorithm and Support Vector Machine	In this paper, the proposed machine learning framework outperforms network intrusion detection systems using a genetic algorithm and support vector machines as two important components of the framework.
M. Bensalem, S. K. Singh	On Detecting and Preventing	Optisystem software to simulate a jamming attack	Artificial Neural Networks,	A machine learning framework was proposed for detecting and

and A. Jukan	Jamming Attacks with Machine Learning in Optical Networks (Bensalem, Singh and Jukan, 2019)	and generate experimental datasets	Support Vector Machine, K-nearest Neighbour, Decision Trees, Naïve Bayes, and Logistic Regression	preventing jamming attacks in optical networks, where it was found that Artificial neural networks performed the best accuracy and time complexity to detect and localise out-of-band power jamming attacks.
Isra'a AbdulNabi∗, Qussai Yaseen	Spam Email Detection Using Deep Learning Techniques (AbdulNabi and Yaseen, 2021)	Spambase data set from the UCI machine learning repository & Spam filter dataset from Kaggle	BiLSTM model, compared with KNN [n-neighbours =3]	Bert contextual word embedding improves the capability of detecting spam emails compared to Keras word embedding.

From the Systematic Literature review:

A more specific search was conducted in this literature review using a specific search technique. From the systematic literature review seen in Table 1, it can be seen that several research papers have identified the use of machine learning algorithms applied to predict/detect malware. Instead of using traditional statistical analysis methods to predict malware, Pan, et al turn to other methods using machine learning and data mining (Pan, et al Tang and Yao, 2020). The paper also proposes to report the results of three algorithms, including Logistic Regression, K-Nearest Neighbours and LightGBM and compare these results to pinpoint the algorithm best suited for predicting the possibility of malware on the Microsoft Malware prediction dataset on Kaggle. In work conducted by Pan et al, a thorough explanation is provided in the steps taken to clean the dataset along with reasons why the dimensionality of the dataset was reduced from 83 to 42 columns using Chi-square testing. The selection of the LightGBM model is supported by the work conducted by Ke et al as it can be concluded that LightGBM 'speeds up the training process of conventional gradient boosting decision trees by up to over 20 times achieving similar accuracy.' Kumar et al use a similar approach to Pan et al by choosing a tree-based algorithm, however, instead of working with LightGBM, Kumar uses the XGBoost algorithm on the EMBER dataset, which consists of 1.1 million entries with a label of malware. Kumars comparison between XGBoost and other classification models shows that the XGBoost algorithm performed the best, achieving 98.2% after hyper tuning specific parameters such as various learning rate values ranging from 0.01 to 0.2 and a n_estimator value of 600 (Kumar and S, 2020). Other machine learning algorithms have also been used in the prediction/detection of malware, which is seen by McLaughin et al., who use deep convolutional neural networks for malware classification. McLaughin explains the various layers required to build the

Deep Convolutional Neural Networks model consisting of an opcode embedding layer and convolutional layers and reports the model's findings. This paper also reports that on the small Raw Dalvik bytecode of an Android application dataset, the convolutional neural network achieved an accuracy of 98%, on a larger dataset, an accuracy of 80% and on a larger dataset, an accuracy of 87%. Looking closer at the results, it can be interpreted that the model performs well on the very large dataset achieving a precision of 0.87and a recall of 0.85 (McLaughlin et al., 2017).

2.6. Research Gap

In recent times it has been reported that UK firms are 'most likely to pay' (Tidy, 2022) ransomware hackers epitomising the threat of cyber-attacks and the need for machine learning models that can predict/detect malware. With around 82% of companies in the UK being likely to pay hackers to retrieve their data and 56% of global companies being victims of cybercriminals, the importance of the research conducted in the previous section is emphatic. The development of machine learning algorithms must be continued to reduce the number of malware cyber-attacks and enable companies to provide an extra barrier of security. An evident research gap has been identified from the research conducted, which is the prediction of malware given several features. The solution being proposed includes the exploration of the Microsoft Malware Prediction dataset found on Kaggle, which will be analysed, cleaned, and pre-processed before using creating machine learning models to predict the probability of malware on a windows machine.

3. Approach

3.1. Introduction

This project will predict a Windows machine's probability of getting infected by various families of malware based on the different properties of that machine using the Microsoft Malware Prediction Dataset from Kaggle. Malware detection is a binary classification. In this project, one refers to the presence of malware and 0 refers to the machine being malware-free. To implement these classifiers, I will use ML algorithms that will be trained using the dataset that has been generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender. The most common ML models derived from the literature review are tree-based models like Light Gradient Boosting Machine (LightGBM), XGBoost, AdaBoost, and CatBoost. Also popularly used classifiers are Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes (NB) and Neural Networks (NN). From my literature review, the popularity of Tree-based machine learning algorithms has grown due to two reasons. The first is the ability of Tree-based algorithms to handle diverse data without the need of pre-processing raw data extensively, and the second is the fast computational Implementation of Tree-based algorithms. These two reasons have been highlighted by the work of Kern et al., who demonstrate diverse, prospective implementations of Tree-based algorithms along with discussions of previously implemented models (Kern, Klausch and Kreuter, 2019).

Neural Networks have also proved to be efficient in clustering and classifying data when trained on a labelled dataset. Thus, the following section of this report will highlight LightGBM, XGBoost and Artificial Neural Networks.

The flowchart of this approach is seen in Figure 2.

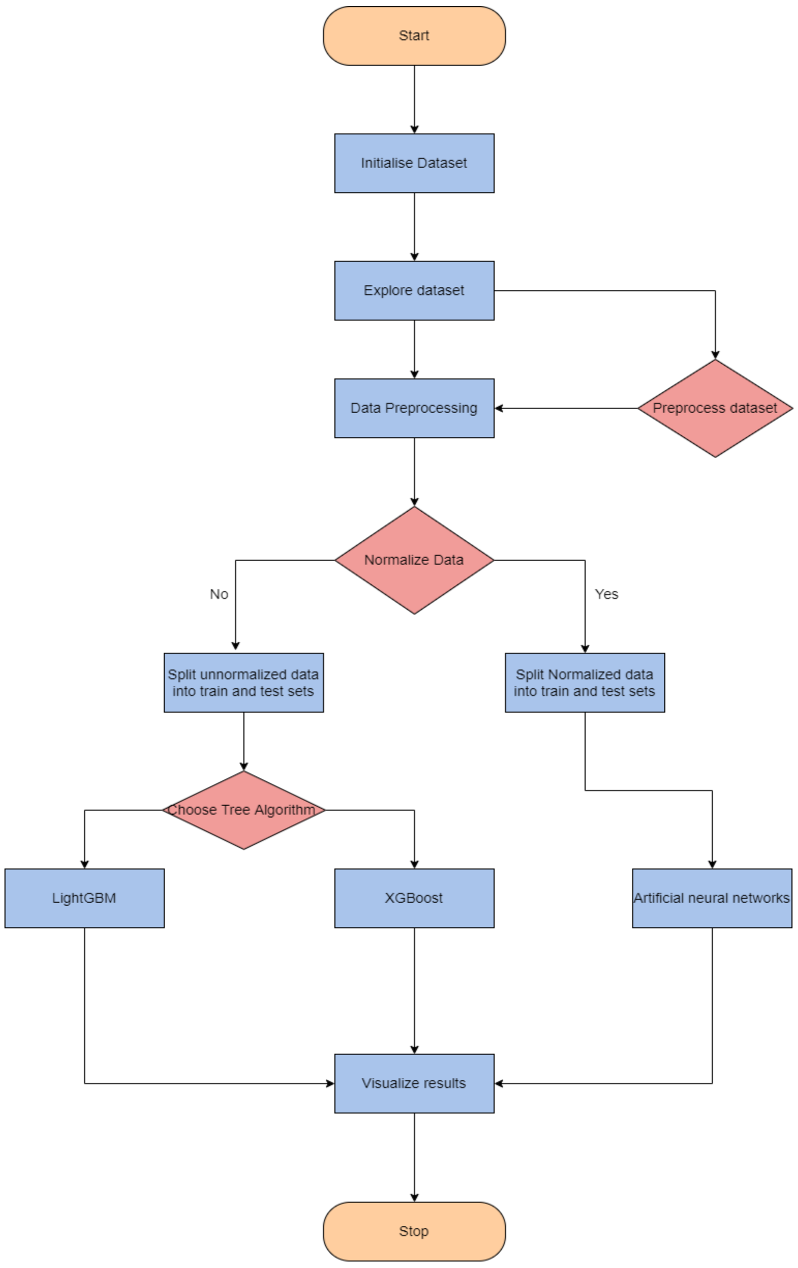


Figure 2 Flow of Implementation

3.2. Ingestion of data

Data ingestion will be done using Pandas, this process includes moving data from the local drive into a Pandas Dataframe structure (phadnispradnya, 2021). The source file is a Comma Separated Values (.csv) file and can be ingested using the pd.read_csv('train.csv') as our dataset is called train.csv. This process is also discussed in detail in the Implementation section 4.1.1.

3.3. Explanatory data analysis

EDA is one of the first steps in implementing a machine learning algorithm. It is an approach for gaining the highest level of insight into the data imported (NIST, 2019). During EDA, the data will be investigated and summarised for its main characteristics. Most of the findings will be represented graphically. It will also find anomalies in the data and understand how the dataset will be modified during prepossessing.

3.4. Data Pre-processing

Date pre-processing involves transforming the raw data from the source file into an understandable and useful format. Once data analysis is complete, pre-processing will be done to make the dataset more complete and efficient to improve the performance of classification models. This process includes data cleaning, which makes sure there is no unnecessary data tampering with the accuracy of the models later.

3.5. Classification Models

3.5.1. LightGBM model

LightGBM (Bachman, 2018) is a gradient boosting framework that is based on a decision tree algorithm. It is an open-source library that can be used for ranking as well as Classification. It splits the tree leaf-wise with the best fit, unlike other boosting algorithms that split the tree depth-wise. LightGBM gives better accuracy than other boosting algorithms because leaf-wise algorithms reduce loss more than level-wise algorithms. It is also more efficient in terms of speed.

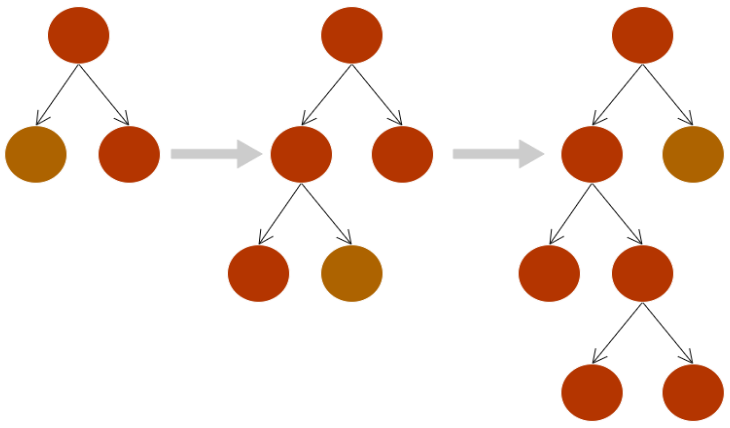


Figure 3 Leaf wise tree growth in LightGBM

There are multiple advantages to using LightGBM, like low memory usage – and replacing continuous values with discrete bins. It is compatible with big datasets and performs well, along with a substantial reduction in training time. Even though leaf-wise splitting can lead to overfitting due to increased complexity, it can be repaired by specifying another 'max-depth' parameter.

3.5.2. XGBoost model

XGBoost stands for eXtreme Gradient Boosting. It is an implementation of a scalable, distributed gradient boosting decision tree (GBDT) designed for better speed and performance (Jason Brownlee, 2016). It accomplishes machine learning tasks like regression, Classification, and ranking (NVIDIA, n.d.). It builds upon the concept of supervised machine learning, which, as discussed before in this report, uses algorithms to find patterns in datasets. This is done by training a model with a labelled dataset and then using the same to predict the labels on a new dataset. XGBoost, unlike LightGBM, follows a level-wise splitting strategy. This means it scans across all gradient values, and with these partial sums at every split, it evaluates the quality of the splits in the training set.

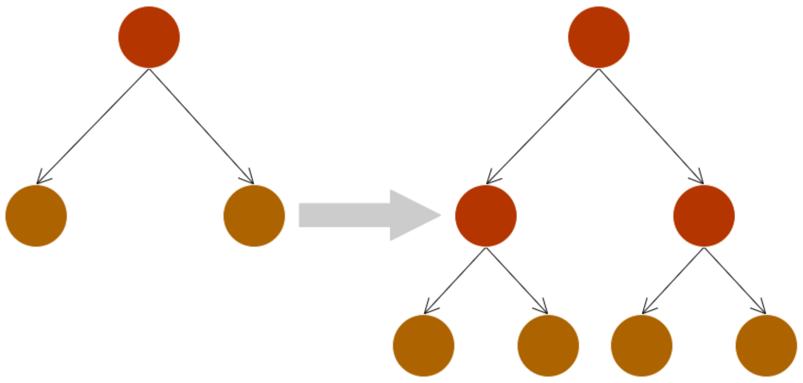


Figure 4 Level wise tree growth in XGBoost

XGBoost also has an extensive list of benefits. This is an open-source development; therefore, machine learning enthusiasts all over the world are constantly contributing to the project. The machine learning models created have a highly efficient combination of prediction performance and execution speed – due to this reason, it appears most frequent on Kaggle as a go-to algorithm for winners of competitions.

3.5.3. Artificial Neural Networks

Neural networks (Nicholson, n.d.) are modelled based on the human brain, they help recognise patterns, interpret data through machine perception and help classify input data. The data classified can be numerical and contained in vectors. In this case, neural networks were chosen because they also extract features that are fed into other algorithms. However, the end goal of using his is to help group unlabelled data based on the similarities between the input data, which is achieved when neural networks have a labelled dataset to train on. Neural Networks consist of artificial network functions, commonly known as parameters, similar to other classification algorithms. These parameters receive multiple inputs and produce one output each. These outputs are passed on as input to the subsequent layers of parameters, and this goes on until each layer of the parameters has been considered and the last parameters have received an input – these last parameters give the final result of our model.

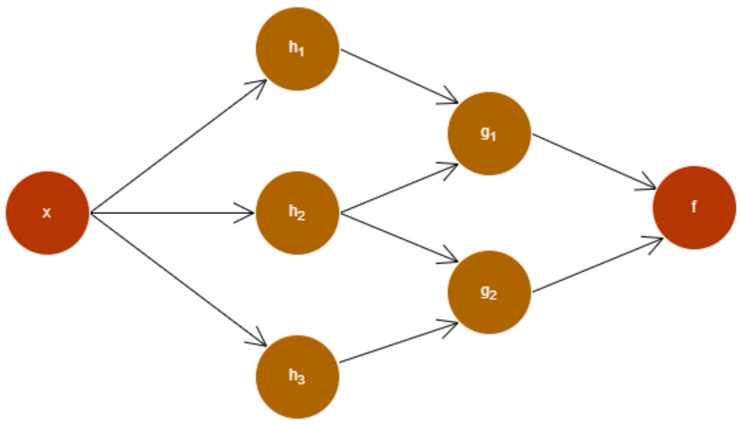


Figure 5 Visual representation of a Neural Network

Figure 5 shows a simple neural network, the initial input is 'x', and it is passed to the first layer of parameters, which are also called neurons. The neurons are represented by h_1, h_2, and h_3 – these functions generate output and pass it on to the second layer represented by g1 and g2. Once the output is calculated from the ‘g’ layer, it is combined to yield the model's final output, 'f'.

3.6. Agile Methodology

The project adheres to an agile methodology. This methodology consists of six key stages which were followed in this project. The problem was defined, and a fundamental understanding was built from the research conducted in Cybersecurity and Machine Learning. Through the conducted research, a thorough plan and direction were given to the project by setting up paths to design and implement the project. A plan was formed outlining the project goals that must be met at the end of the process. An agile methodology allows these checkpoints to be updated depending on any extra features that need to be implemented. The planning stage highlighted the technological equipment which will be used for the machine learning models and the time frame for the project to be completed. This stage includes milestones set specifically for this project.

The design stage is another vital stage that has been depicted in the diagram above. This stage includes defining the approach that will be a foundation for the project and help during the Implementation.

The implementation stage uses the fundamentals gathered from the planning, literature review, and design stages to code the proposed models, hyper tune the parameters, and then evaluate them.

The testing stage checks that the Implementation of the system has been performed correctly, ensuring all the functionalities work well.

The results of the tests are then analysed, and a conclusion is drawn as to which model performs the best in terms of accuracy, computational power, and time.

4. Implementation

4.1. Design and Implementation

4.1.1. Dataset

The dataset (Kaggle, 2018) used for this study was acquired from Kaggle. Microsoft provided an extraordinary new dataset to encourage open-source collaboration for predicting malware. This dataset follows its malware challenge in 2013 (Kaggle, 2013). However, this telemetry dataset generated by combining heartbeat and threat reports collected from Microsoft's endpoint protection solution – Windows Defender, contains properties of a machine and infections. This dataset was, of course, sampled, keeping in mind various business limitations in terms of privacy and during the machine running. It was noted that Malware Detection, in its essence, is a time-series problem which means that observations close to each other in time are similar compared to observations far away. However, this time series problem is made complicated when new machines are introduced, when machines are not always online or when machines receive patches and are updated with new operating systems. Microsoft also highlighted that the dataset was roughly split by time and does not represent its customers because there had to be a large number of malware machines sampled to balance the dataset.

The train.csv dataset was downloaded and used as labelled information to train the three models discussed earlier. Each row in the datasets corresponds to a machine with the primary key 'MachineIdentifier', which, as the name suggests, is a unique machine ID. Another key column is 'HasDetections' which is the ground truth – it is a binary value for each machine, indicated by 1 or 0 if malware was detected on the machine or not respectively.

The first step was to start the data processing, for which I imported the dataset into a Jupyter notebook. The implementation code used to read the **train.csv** file and convert it into a dataframe (df) was using a python library 'Pandas'.

Once loaded, it was noted that this data contained 83 columns, also known as features and 8921483 rows. Keeping computational time and computational power in mind, the dataset was scaled down to 1 million rows and stored as **shortSet.csv**.

4.1.2. Exploratory Data Analysis

A sum of 0s and 1s was calculated to check if the dataset was balanced. The total number of machines with malware was 499813, and 500187 machines had no malware present. As we can see from this information, the training dataset contains equal amounts of positive and negative detections, concluding that the dataset is balanced. A balanced dataset is vital so that the model is not biassed towards a specific value of 'HasDetections' and can classify the data accurately.

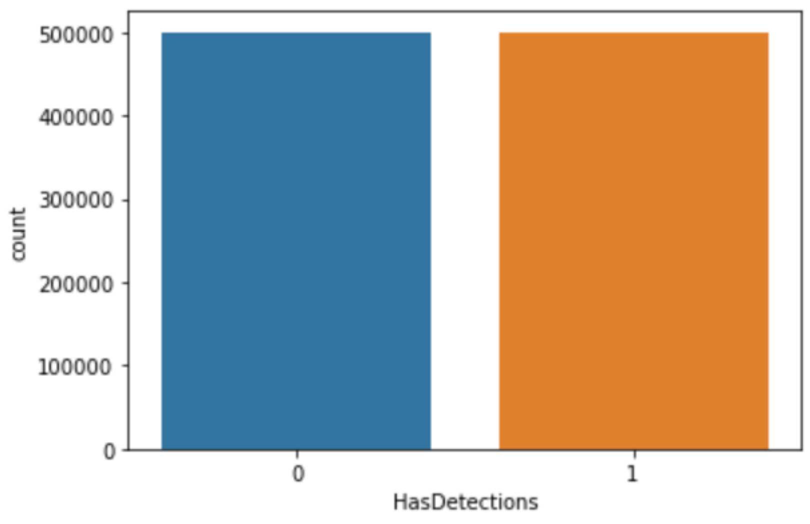


Figure 6 Bar Graph for 'HasDetections'

Each column in this dataset represents a property of the machine, and these are divided into three types: binary, numerical and categorical. To find how many columns belong in each category, conditions were added for each of the categories to define them.

Once defined, I used the python 'Plotly' library to create a visualisation in the form of a pie chart.

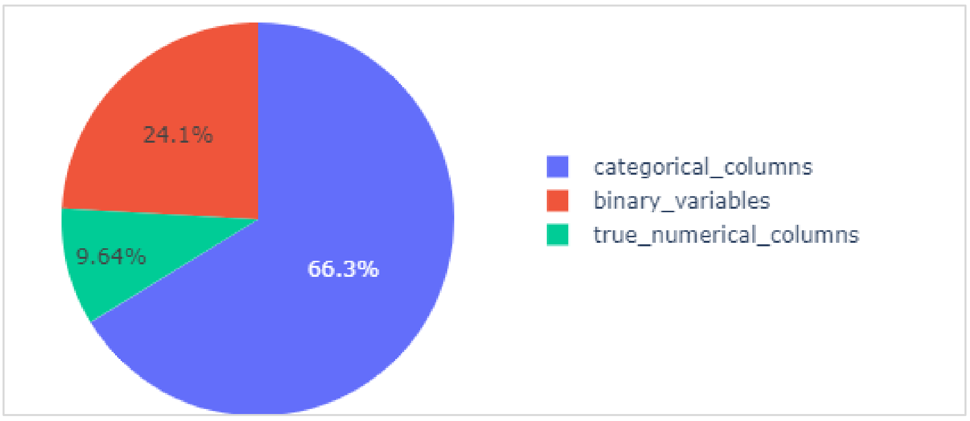


Figure 7 Pie Chart for column category

This graph confirmed that the dataset contains 55 categorical columns, 20 binary columns, and eight numerical columns. From this chart, we know that majority of the columns are categorical, and this is important to know because, in order to build a machine learning model at a later stage, if we have categorical columns, we will have to implement encoding so that the machine learning algorithms can understand that it is a categorical variable and use them accordingly.

Following this, it is essential to calculate how many unique values are present in each categorical column, this is called cardinality. For binary columns, the cardinality is two, as it represents unique values in any given volume. To find this for categorical columns, I plotted a bar graph using plotly. As you can see in Figure 8, the top few columns have big bars, and the ones in the bottom are almost negligible – this is because the highest value on the x-axis is 60 thousand. The reason for such high unique values, or to say cardinality count in the top variables, is that they are some sort of **identifier**. Now, these categorical variables that have high cardinality will be encoded.

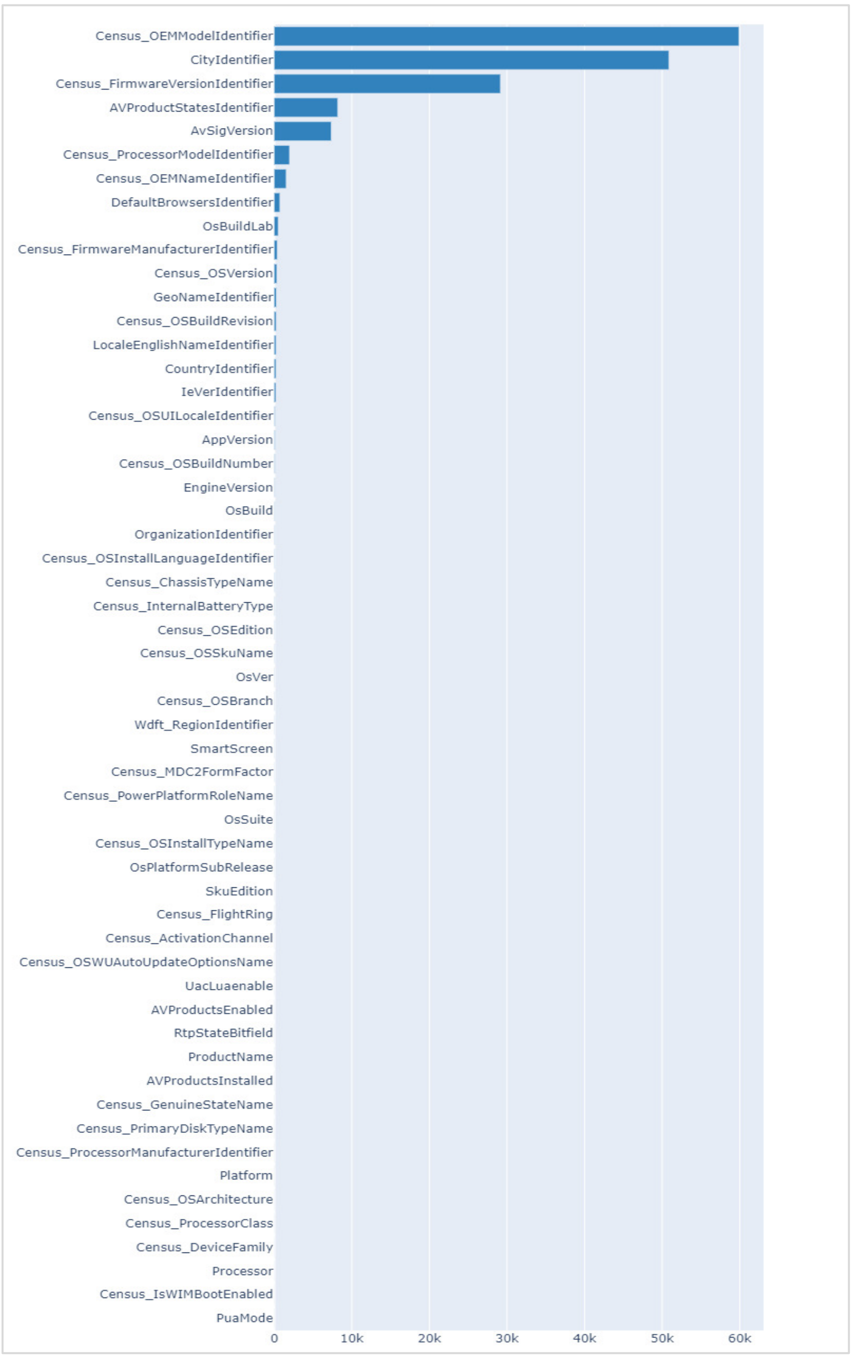


Figure 8 Categorical Cardinality

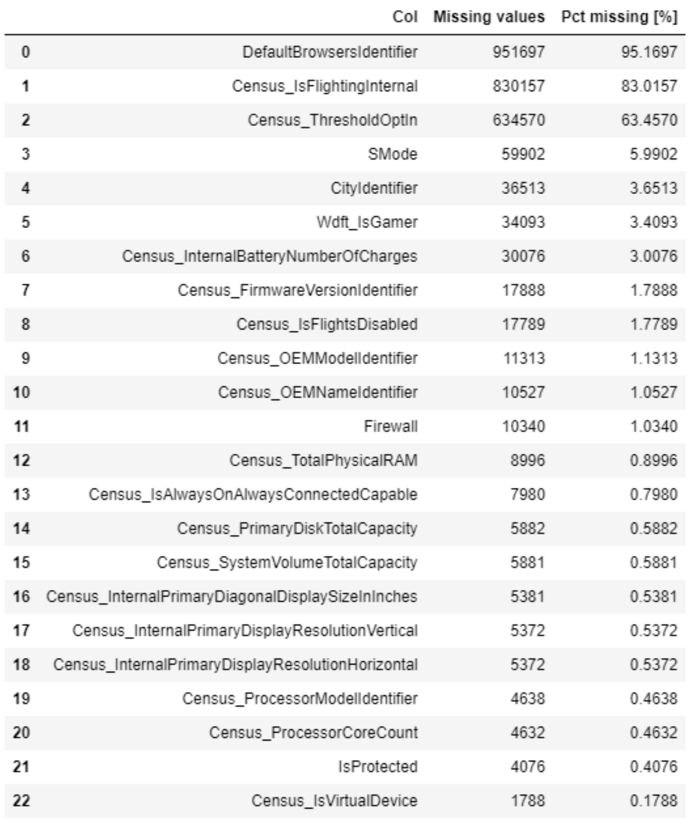
I decided to pick out the first seven categorical columns with the highest cardinality, i.e., 'Census_OEMModelIdentifier', 'CityIdentifier', 'Census_FirmwareVersionIdentifier', 'AvSigVersion', 'Census_ProcessorModelIdentifier', 'Census_OEMNameIdentifier', and 'DefaultBrowsersIdentifier' for feature engineering.

4.1.3. Feature Engineering

Feature Engineering is an integral step in the usage of machine learning, and it is the practice of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done successfully and accurately, it helps increase the prediction power of the algorithms by creating features from raw data. This is always implemented before defining the model.

There are multiple categorical columns in almost every dataset where the variables are stored as strings (text values), whereas machine learning is based on mathematical equations. If we were to keep categorical columns, it might cause problems. There are certainly some algorithms that support categorical data, but for the purpose of uniformity, I decided to encode the ones with the highest cardinality. There are multiple encoding techniques like nominal encoding, where the order of the data doesn't matter; ordinal encoding, where the order of the data matters; and frequency encoding, where we utilise the frequency of categories as labels in case the frequency is related to the target variable it will help the model understand better. I decided to use frequency encoding, and in this, if the columns are related to the target variable 'HasDetections', it will make the machine learning algorithms more efficient.

Once frequency encoding was complete, the next task was to clean the dataset. In order to do this, I first found the amount of null value by percentage in the features. There were 23 features with NaN values.



	Col	Missing values	Pct missing [%]
0	DefaultBrowsersIdentifier	951697	95.1697
1	Census_IsFlightingInternal	830157	83.0157
2	Census_ThresholdOptIn	634570	63.4570
3	SMode	59902	5.9902
4	CityIdentifier	36513	3.6513
5	Wdft_IsGamer	34093	3.4093
6	Census_InternalBatteryNumberOfCharges	30076	3.0076
7	Census_FirmwareVersionIdentifier	17888	1.7888
8	Census_IsFlightsDisabled	17789	1.7789
9	Census_OEMModelIdentifier	11313	1.1313
10	Census_OEMNameIdentifier	10527	1.0527
11	Firewall	10340	1.0340
12	Census_TotalPhysicalRAM	8996	0.8996
13	Census_IsAlwaysOnAlwaysConnectedCapable	7980	0.7980
14	Census_PrimaryDiskTotalCapacity	5882	0.5882
15	Census_SystemVolumeTotalCapacity	5881	0.5881
16	Census_InternalPrimaryDiagonalDisplaySizeInInches	5381	0.5381
17	Census_InternalPrimaryDisplayResolutionVertical	5372	0.5372
18	Census_InternalPrimaryDisplayResolutionHorizontal	5372	0.5372
19	Census_ProcessorModelIdentifier	4638	0.4638
20	Census_ProcessorCoreCount	4632	0.4632
21	IsProtected	4076	0.4076
22	Census_IsVirtualDevice	1788	0.1788

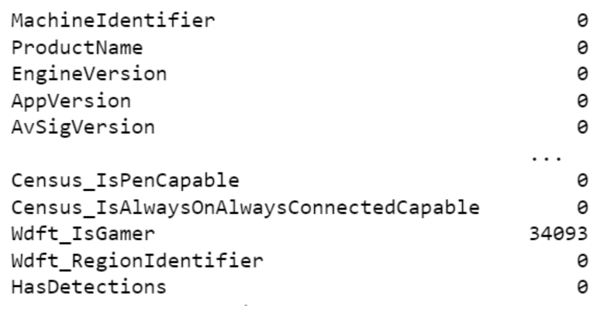
Figure 9 Percentage of NaN values

I dropped the columns consisting of more than 50% of NaN data, i.e 'DefaultBrowserIdentifier', 'Census_IsFlightingInternal' and 'Census_ThresholdOptin'.

Even though LightGBM and XGBoost can handle NaN values, while implementing a neural network, I learned that training a neural network with missing data can lead to the output layer containing Nan values, and the model will end up with the wrong Classification.

4.1.4. Data Pre-processing

In order to remediate this, I filled the NaN values of each column with the respective averages of the data by using the pandas command **fillna()** and **mean()**. Following this, I checked the sum of null values in each feature to ensure that there were zero NaN values.



```
MachineIdentifier                              0
ProductName                                    0
EngineVersion                                  0
AppVersion                                     0
AvSigVersion                                   0
                                             ...
Census_IsPenCapable                            0
Census_IsAlwaysOnAlwaysConnectedCapable        0
Wdft_IsGamer                               34093
Wdft_RegionIdentifier                          0
HasDetections                                  0
```

Figure 10 Number of NaN values after fill.na() function

At this stage, there was only one feature that had 34,093 NaN values – 'Wdft_IsGamer'. To remediate this, I used the **describe()** function to understand this feature. This feature had a minimum value of 0 and a maximum of 1. Hence **min()** was used to replace all NaN values with 0.

In conclusion, after the exploratory data analysis, pre-processing, and feature engineering, I was able to conclude that the dataset is balanced and clean. For implementing the three machine learning models, the dataset consisted of 80 columns with no NaN values.

4.2. Classifiers

4.2.1. Introduction

This project includes the Implementation of four different popular machine learning models, which have been discussed prior to the design of the project. The models include LightGBM, XGBoost, Artificial Neural Network and Support Vector Machine; the two tree-based models have been implemented using their own individual libraries. The Artificial Neural Network has been developed using Tensorflow, and the Support Vector Machine algorithm has been developed using sklearn. The Implementation of these four models aims to evaluate their individual performance on the Microsoft Malware prediction dataset post the data pre-processing and perform a comparison regarding accuracy and computation time.

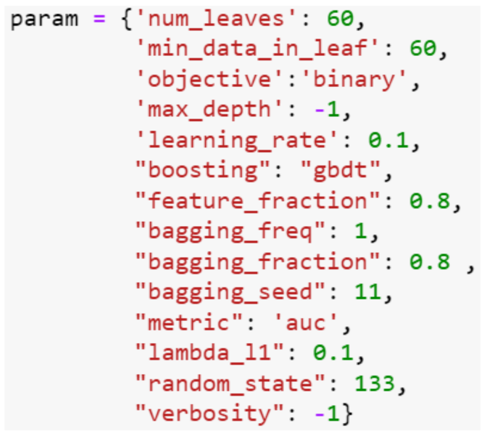
This section will discuss the implementation of the three models and an explanation of the parameters that have been selected in the development.

4.2.2. Implementation of LightGBM

LightGBM was first defined in 2017 in a paper titled' LightGBM: A Highly Efficient Gradient Boosting Decision Tree' (Guolin Ke et al., 2017). This paper introduced two ideas: GOSS – Gradient-based One-Side Sampling and EFB – Exclusive Feature Building. GOSS is an amendment to the standard gradient boosting method that focuses on the training examples that result in a larger gradient. This, in turn, speeds up the learning process and reduces computational complexity.

The LightGBM algorithm based on decision trees is different from other regular decision tree algorithms as the algorithm splits the tree leaf-wise rather than other decision tree algorithms that are split tree level-wise. In the Implementation of this project-specific, parameters have been tuned to increase the accuracy and efficiency of the model.

Some of the parameters that have been identified and tuned are the following: 'num_leaves' has been set to 60. The initial value for this parameter was 31 and signifies the complexity of the model by initialising the value to 60, the maximum number of nodes per tree has been increased. By increasing the number of leaves, this parameter could be beneficial for in terms of an increase in accuracy but could also lead to overfitting problems. Similarly, the 'min_data_in_leaf' has been set to 60, the initial value for this parameter was 20. As the name suggests, this parameter refers to the minimal number of data on the leaf, and it is used to prevent over-fitting. The optimal value for this parameter is dependent on the value of num_leaves, to avoid any over-fitting that might occur due to the increase in the number of maximum nodes, the 'min_data_in_leaf' parameter has been increased. The objective function has been set to binary as this problem is a binary classification problem identifying whether certain machines contain malware or not. Other parameters that have been tailored include 'feature_fraction', which selects a subset of features on each iteration before training each tree. The default value for this parameter is set to 1.0; however, in this LightGBM model, the parameter is set to 0.8. By setting this parameter to 0.8, the LightGBM model will randomly select 80% of the features at the beginning of the construction of each tree, leading to a decrease in the total number of splits that are required to be evaluated to add each node. Another benefit of tuning this parameter is that it can speed up the training and deal with overfitting. The 'bagging_freq' and 'bagging_fraction' are also two interlinked parameters that have been tuned to benefit the model. The values of these two parameters have been set to 1.0 and 0.8, respectively; by tuning these two parameters, the training data is resampled every iteration, and samples are drawn from 80% of the training dataset.



```
param = {'num_leaves': 60,
         'min_data_in_leaf': 60,
         'objective':'binary',
         'max_depth': -1,
         'learning_rate': 0.1,
         "boosting": "gbdt",
         "feature_fraction": 0.8,
         "bagging_freq": 1,
         "bagging_fraction": 0.8 ,
         "bagging_seed": 11,
         "metric": 'auc',
         "lambda_l1": 0.1,
         "random_state": 133,
         "verbosity": -1}
```

Figure 11 Parameters for LightGBM

To maximise the use of the training and testing dataset and the Implementation of the LightGBM model, K-Fold Cross Validation has been implemented to assess the model's performance. In the Implementation of this procedure, the dataset is shuffled randomly and then split into k folds (bins or groups); in the case of the current implementation, k has been chosen to be 6. For each unique group, a specific group is taken out and initialised as the test dataset, and the remaining groups are the training dataset. The model is trained on the training set and evaluated on the test dataset. In the next iteration, a different group is chosen to be the test dataset, and the remaining groups of data are held to be the training dataset which is then again evaluated. This process is repeated for each iteration until each group/bin has become the test dataset.

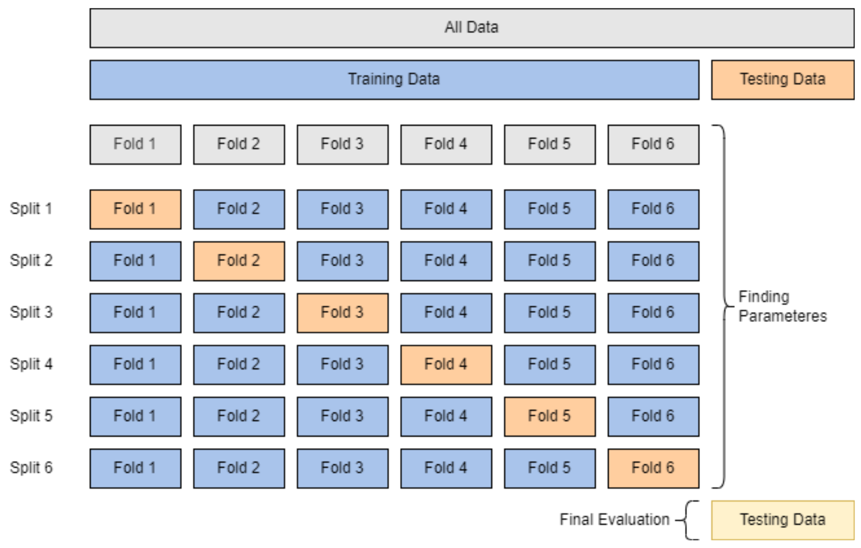


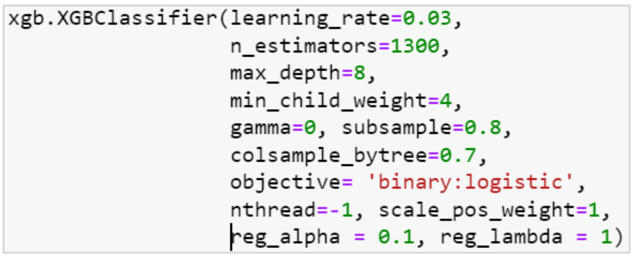
Figure 12 K-Fold Procedure

The early stopping method was also implemented during the training process. Early stopping stops the training process if the model performance does not improve for several iterations.

4.2.3. Implementation of XGBoost

The second tree-based algorithm that has been selected for this classification problem is the ‘highly effective tree-boosting algorithm’ – XGBoost (Chen and Guestrin, 2016). This is an open-source and scalable machine learning system for tree boosting. To implement this algorithm, the XGBoost optimised distributed gradient boosting library has been used, and the parameters for this algorithm have been tuned.

The XGBClassifier has been initialised with a 'learning_rate' of 0.03, and the 'n_estimators' value has been initialised to 1300 from its initial value of 100. The 'n_estimator' value corresponds to the value of the number of trees in the XGBoost model that is being implemented. The 'max_depth' of the tree has been tailored to 8 instead of its default value of 6; this change represents the size of the trees. The 'min_child_weight' has been increased from 1 to 4 in the training of this model on the Malware dataset; by increasing the value of the 'min_child_weight,' the model will find it more difficult to create new nodes in a tree and will reduce the complexity of the tree leading to the model to less likely overfit. Other parameters that have been tuned can be seen in Figure 13 below.

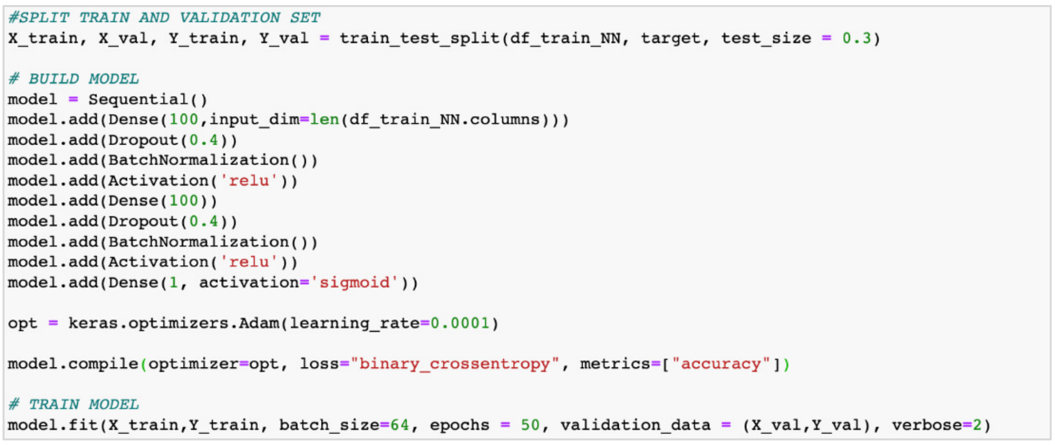


```
xgb.XGBClassifier(learning_rate=0.03,
                  n_estimators=1300,
                  max_depth=8,
                  min_child_weight=4,
                  gamma=0, subsample=0.8,
                  colsample_bytree=0.7,
                  objective= 'binary:logistic',
                  nthread=-1, scale_pos_weight=1,
                  reg_alpha = 0.1, reg_lambda = 1)
```

Figure 13 Parameters for XGBoost

4.2.4. Implementation of Neural network

The third algorithm implemented to solve this classification problem is Artificial Neural Networks (ANN). Artificial Neural Networks architectures are inspired by the ‘sophisticated functionality of human brains.’ ANNs consist of neurons and layers and hidden layers, which have grown in popularity in recent times as various domains have adopted and implemented this for ‘image processing and language processing – this is because it can automatically learn and abstract features’ (Tobiyama et al., 2016). This section will go through and explain the construction of the Neural Network, indicating the different layers of the architecture.



```
#SPLIT TRAIN AND VALIDATION SET
X_train, X_val, Y_train, Y_val = train_test_split(df_train_NN, target, test_size = 0.3)

# BUILD MODEL
model = Sequential()
model.add(Dense(100,input_dim=len(df_train_NN.columns)))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))

opt = keras.optimizers.Adam(learning_rate=0.0001)

model.compile(optimizer=opt, loss="binary_crossentropy", metrics=["accuracy"])

# TRAIN MODEL
model.fit(X_train,Y_train, batch_size=64, epochs = 50, validation_data = (X_val,Y_val), verbose=2)
```

Figure 14 ANN Code Cell on Jupyter Notebook

To construct the architecture of this neural network Keras and TensorFlow had been identified to be two important libraries when creating the Artificial Neural Network Layer. First, using the 'sklearn train_test_split,' the pre-processed dataset was split into a training and test set in a 70:30 ratio, respectively. Next, a sequential model was defined, which allows multiple layers to be stacked over each other, with each layer consisting of one input tensor and one output tensor. The first layer initialised in the sequential model is the Dense Layer which is deeply connected to the prior layer, in this network, the Dense Layer consists of 100 neurons and takes the input dimensions of the training dataset.

The Dense layer consists of several neurons where each neuron receives an input from the previous layers of neurons. In this case, the dense layer is the first layer of the sequential model. The next layer is the Dropout layer used to 'randomly set the outgoing edges of hidden units to 0 at each update of the training phase’ (Keras, n.d.). Deep neural networks are prone to a fundamental problem of overfitting and heavy computational power and expense. To avoid these fundamental problems, the Dropout layer has been initialised with a probability of 0.4, indicating that there is a 40% chance the output neuron will be changed to 0. The batch normalisation layer is added to normalise the inputs of the layer by applying transformations that maintain the mean output close to 0 and standard deviation close to 1.

The Activation layer is the final unique layer used for this model, which applies the activation function chosen and is an essential component of any architecture as this layer computes the weighted sum of inputs and bias’s that can be used to assess if the neuron can be fired or not (Chigozie et al., 2018). In implementing this neural network architecture, the 'ReLU’ activation function has been used along with the 'sigmoid' activation function. Both visual representations can be seen in Figures 15 and 16 below.

ReLU: rectified linear activation function is a linear function that will output the input directly if it is positive, otherwise, it will output zero.

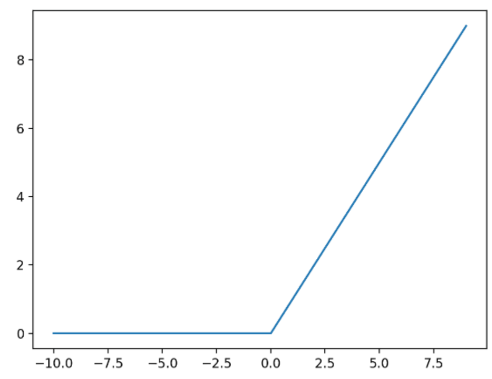


Figure 15 ReLU function graph

Sigmoid: this activation function guarantees that the output of this unit will always be between 0 and 1.

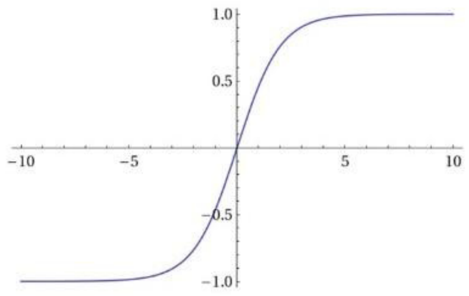


Figure 16 Sigmoid function graph

Finally, the model is compiled with a learning rate of 0.0001, a loss function of binary_crossentropy and an Adam optimiser.

5. Testing

This section demonstrates that the project has been developed as intended and is able to answer the research question proposed. The tests have been carried out using Python v3 on Jupyter Notebook v6.4.5 running on windows 10 OS.

Table 2 Test Case - 1

Test Id: 01	**Test Purpose**: Loading dataset from the source		
Preconditions: download dataset train.csv from Kaggle, import Pandas			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Input the name of the file in the cell, read CSV and convert it into dataframe	When loaded, run the file to make sure all 83 columns are shown	Pass
Comments: Original file train.csv was shortened to shortSet.csv with 1 million rows.			
Related Tests: n/a			

Table 3 Test Case - 2

Test Id: 02	**Test Purpose**: Exploratory Data Analysis procedure to understand and visualise the dataset		
Preconditions: shortSet.csv is loaded, import Plotly			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Check if the dataset is balanced by calculating the sum of 0s and 1s in the HasDetections column, then create a bar graph.	Machines with malware: 499813 Machines without malware: 500187	Pass
2	Create a pie chart for three types of data: binary, categorical and numerical.	categorical columns: 55 binary columns: 20 numerical columns: 8	Pass
3	Visualise cardinality by checking the number of unique values in each column.	Bar graph with 55 columns containing unique values	Pass
Comments: Feature engineering for columns with high cardinality to be performed			
Related Tests: n/a			

Table 4 Test Case - 3

Test Id: 03	**Test Purpose**: Feature Engineering and cleaning the dataset		
Preconditions: List of features with high cardinality			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Implement frequency encoding on seven columns with high cardinality	Encoded with reduced cardinality	Pass
2	Find the amount of NaN values by percentage in each column	Twenty-three features with NaN values, three features with more than 50% NaN values.	Pass
3	Delete the columns with over 50% NaN values	Three columns deleted	Pass
Comments: NaN values in the remaining 20 columns will have to be replaced			
Related Tests: 2			

Table 5 Test Case - 4

Test Id: 04	**Test Purpose**: Data Pre-processing		
Preconditions: 20 columns with less than 50% NaN values			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Taking care of missing data in each column with the respective mean of the data	Only 19/20 columns updated	Fail
Comments: For the remaining column, instead of mean, the min was used to replace NaN values as it was a binary column			
Related Tests: 3			

Table 6 Test Case - 5

Test Id: 05	**Test Purpose**: Coding, Training and Testing LightGBM		
Preconditions: Dataset loaded and pre-processed after running Test#4			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Running all the cells for the LightGBM model with hyper tuned parameters, including k-fold, cross-validation, and confusion matrix	All cells executed with no errors	Pass
Comments: n/a			
Related Tests:			

Table 7 Test Case - 6

Test Id: 06	**Test Purpose**: Coding, Training and Testing XGBoost		
Preconditions: Dataset loaded and pre-processed after running Test#4			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Running all the cells for the XGBoost model with hyper tuned parameters, cross-validation, and confusion matrix	All cells executed with no errors	Pass
Comments: n/a			
Related Tests:			

Table 8 Test Case - 7

Test Id: 07	**Test Purpose**: Coding, Training and Testing Neural Network		
Preconditions: Dataset loaded and pre-processed after running Test#4			
Step No.	**Procedure**	**Response**	**Pass/Fail**
1	Running all the cells for the Neural Net model with hyper tuned parameters, cross-validation, and confusion matrix	All cells executed with no errors	Pass
Comments: n/a			
Related Tests:			

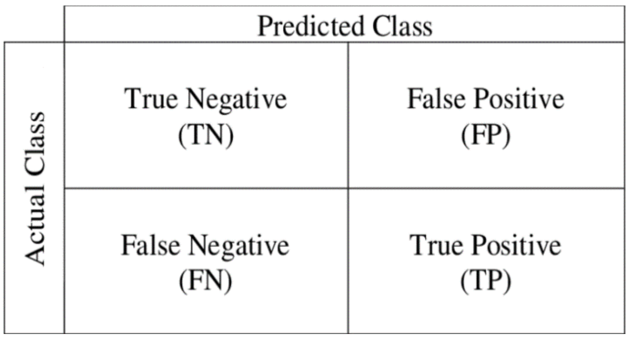
6. Results and Evaluation

This section includes the evaluation technique used and the results for each model.

6.1. Confusion Matrix

Evaluation is a key part of the machine learning process. A confusion matrix table will be used to compare and analyse which classification algorithm in this project had the best performance and is most suitable for malware classification. This indicates how successful the classification algorithm is by summarising the results as four outcomes – TP, TN, FP, and FN (Jayaswal, 2020).

Table 9 Confusion Matrix



	Predicted Class	
Actual Class	True Negative (TN)	False Positive (FP)
	False Negative (FN)	True Positive (TP)

- TP: True Positive: Refers to when the model correctly classifies a positive class as positive.

- TN: True Negative: Refers to when the model correctly classifies a negative class as negative.

- FP: False Positive: Refers to when the model incorrectly classifies a negative class as positive.

- FN: False Negative: Refers to when the model incorrectly classifies a positive class as negative.

This indicates that a good performance indicator would be a higher number of TP and TN values. The confusion matrix accurately indicates the model's classification results and also helps identify other performance indicators like accuracy, precision, recall, and F1 score. These indicators, along with AUC (Area under Curve), help measure the classifiers' overall performance. Each performance indicator is discussed below.

Accuracy: Accuracy refers to the success ratio of the model and is calculated by the formula below. It is the ratio of correctly classified classes to the total Classification of classes.

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

Precision: Precision refers to the number of correct predictive values that the model calculates. It is the ratio of correctly classified predictions to the sum of total classes classified as positive. The formula below is used to evaluate precision.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Recall refers to the number of false negatives calculated by the model. It is the opposite of precision. Recall is calculated by the formula below. It is the ratio of correctly classifies classes to the sum of positive outcomes.

$$Recall = \frac{TP}{TP + FN}$$

F1 score: F1 score is also called F-measure it is the average overall accuracy, i.e. the equilibrium between precision and recall. It takes both false positives and false negatives, and it can be calculated using the formal is below.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

AUC: AUC refers to the Area under the ROC Curve. ROC refers to the graph representing the model's performance at all levels. The ROC curve plots TP rate with respect to FP rate and emphasises the classifier's sensitivity. More Area under the ROC corresponds to a better classifier.

6.1.1. LightGBM Performance

Using the above parameters, the LightGBM model was compared to two other machine learning models on the pre-processed dataset. Its performance can be seen in Figure 18 below. The LightGBM model achieved an accuracy of 66% with a precision and recall of 0.66, respectively. The confusion matrix represents the TP, TN, FP, and FN defined in the previous subsection 6.1 and showed the percentage values of each of the evaluation components. For a large dataset, which is high in dimensions, it can be argued that the accuracy achieved is expected, especially after hyper tuning the parameters as stated in Section 4.2.2. It is important to reiterate some of the key parameters that had been tuned, including the learning rate of 0.1, num_leaves which was set to 60 and num_boost_round set to 1000. The LightGBM result reports an AUC result as 0.721591, which was stopped using an early_stopping_rounds parameter set to 1000. At around 785, the training of the LightGBM model was stopped due to the loss on the validation dataset increasing.

Figure 20 shows the results of the k-fold cross validation implementation results. It is shown that the fourth fold provides the model with the highest accuracy of 0.66266 and an average accuracy of the k-fold cross-validation of 0.65937.

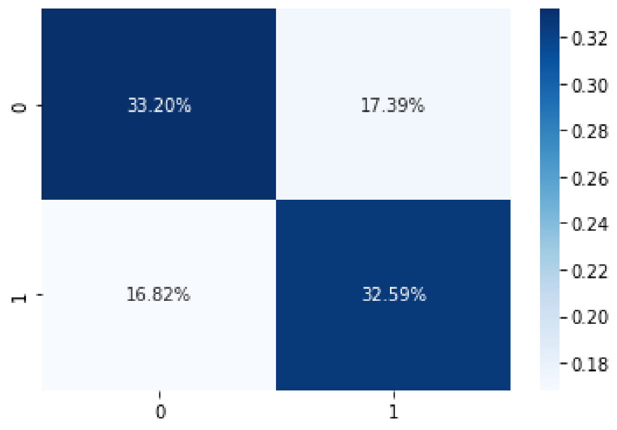
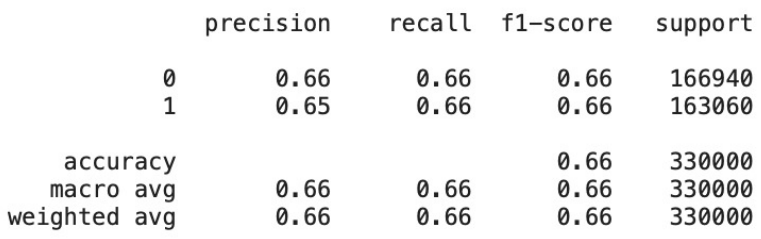
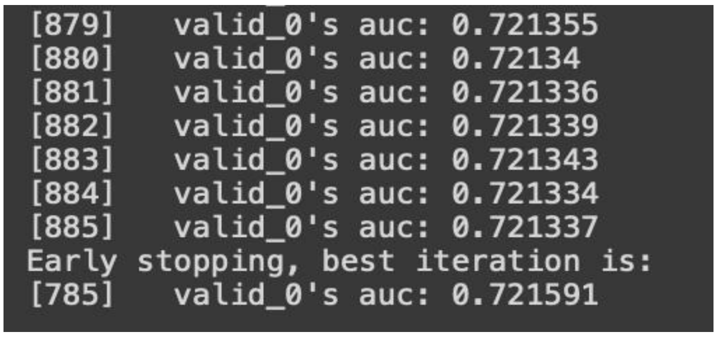


Figure 17 Confusion Matrix for LightGBM



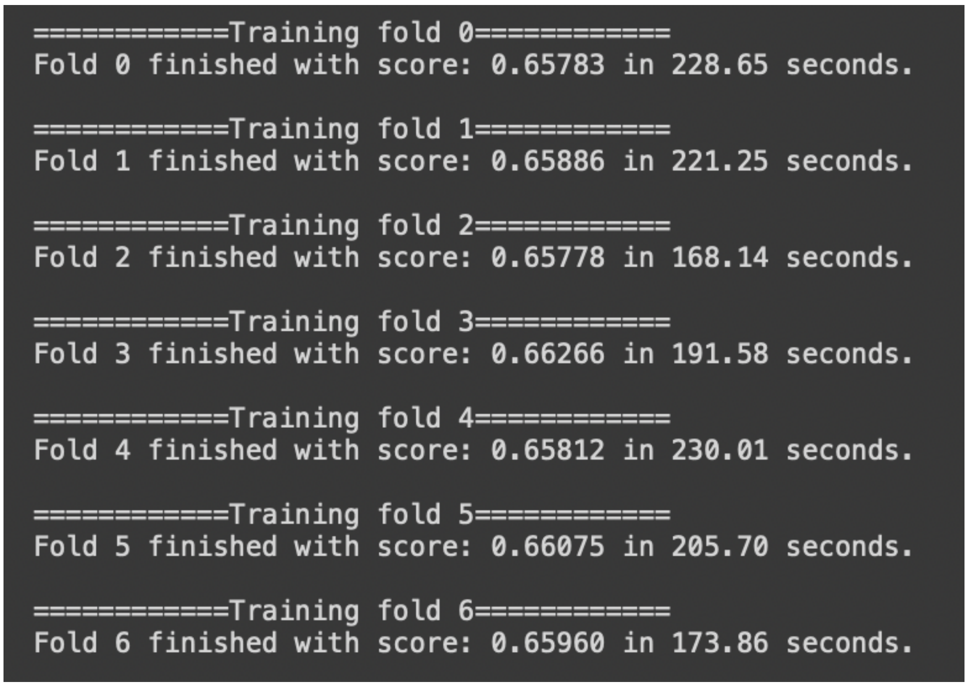
```
              precision    recall  f1-score   support

           0       0.66      0.66      0.66    166940
           1       0.65      0.66      0.66    163060

    accuracy                           0.66    330000
   macro avg       0.66      0.66      0.66    330000
weighted avg       0.66      0.66      0.66    330000
```

Figure 18 Classification Report for LightGBM



```
[879]	valid_0's auc: 0.721355
[880]	valid_0's auc: 0.72134
[881]	valid_0's auc: 0.721336
[882]	valid_0's auc: 0.721339
[883]	valid_0's auc: 0.721343
[884]	valid_0's auc: 0.721334
[885]	valid_0's auc: 0.721337
Early stopping, best iteration is:
[785]	valid_0's auc: 0.721591
```

Figure 19 Early Stopping Function LightGBM



```
============Training fold 0============
Fold 0 finished with score: 0.65783 in 228.65 seconds.

============Training fold 1============
Fold 1 finished with score: 0.65886 in 221.25 seconds.

============Training fold 2============
Fold 2 finished with score: 0.65778 in 168.14 seconds.

============Training fold 3============
Fold 3 finished with score: 0.66266 in 191.58 seconds.

============Training fold 4============
Fold 4 finished with score: 0.65812 in 230.01 seconds.

============Training fold 5============
Fold 5 finished with score: 0.66075 in 205.70 seconds.

============Training fold 6============
Fold 6 finished with score: 0.65960 in 173.86 seconds.
```

Figure 20 Results of the k-fold cross-validation

6.1.2. XGBoost Performance

The next algorithm being evaluated is the XGBoost model, which is another tree-based algorithm explained above. The XGBoost model performed similarly to the LightGBM model achieving 66% accuracy on the pre-processed dataset, which is shown by the classification report in Figure 22. The confusion matrix represents the TP, TN, FP, and FN values which have been defined in the previous subsection and shows the percentage values of each of the evaluation components. The precision, recall and f1-score all come out to a score of 0.66, respectively. This score was reached by hyper tuning key parameters such as the learning_rate, n_estimators and the depth of the tree. The TP value of 32.59% represents the correctly predicted machines that have malware, whereas the TN value of 33.26% represents the correctly predicted machines that do not have malware. These values are very similar to the confusion matrix of the LightGBM model that was shown in Figure 17. This shows that the decision trees performed identically, and either could be chosen as the best machine learning algorithm for this problem set. However, another evaluation factor that can compare both of these models is the computational time taken for these models to run, which has been explained and depicted in Section 6.2.

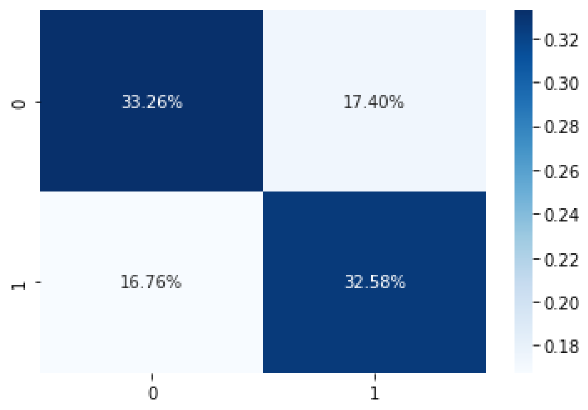


Figure 21 Confusion Matrix for XGBoost

```
              precision    recall  f1-score   support

           0       0.66      0.66      0.66    167176
           1       0.65      0.66      0.66    162824

    accuracy                           0.66    330000
   macro avg       0.66      0.66      0.66    330000
weighted avg       0.66      0.66      0.66    330000
```

Figure 22 Classification Report for XGBoost

6.1.3. Neural Networks Performance

As explained in Section 3.5.3, an artificial neural network was constructed using dense layers, dropout layers and batch normalisation layers with two different activation layers in the architecture. The architecture uses the ReLu and the sigmoid activation functions to decide if the neurons require activation or not. Figure 23 below represents the neural network's learning curves depicting the accuracy and loss of the model. The neural network achieved an accuracy score of 52% on the training and validation set with a loss on 0.6909 and 0.6902 on the training and validation set, respectively. This shows that the neural network has underperformed significantly compared to the previous two tree-based algorithms. However, these results do make sense as artificial neural networks face problems when using a dataset with high dimensionality due to the curse of dimensionality, which is very common in machine learning. It can be argued that a dataset with fewer dimensions can possibly improve the neural network's performance to an extent where it might outperform the performance of decision trees.

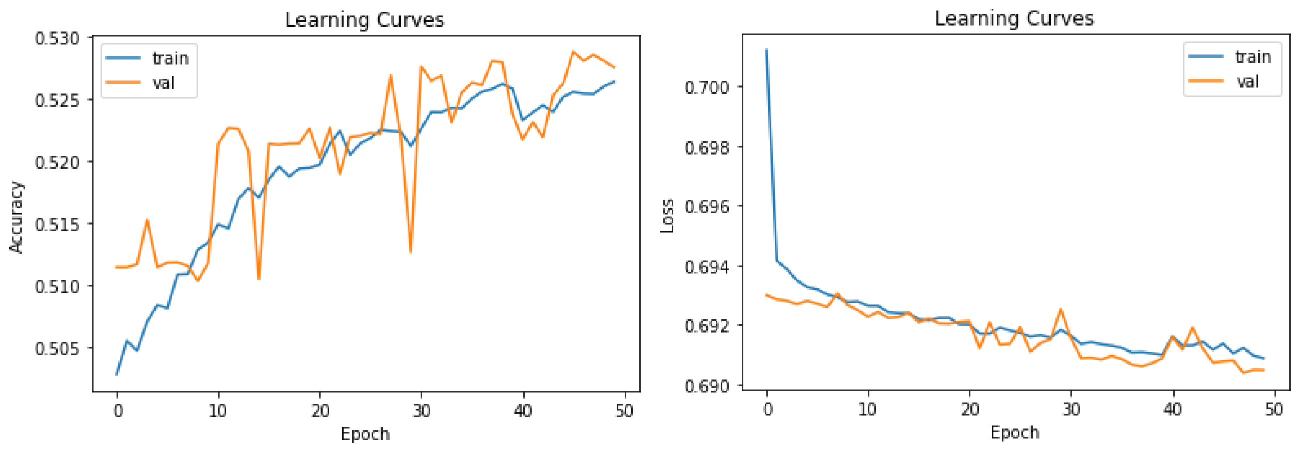


Figure 23 Learning Curves: Accuracy and Loss of the Model

6.2. Comparison of models

After obtaining the individual results for each model, the table below has been formulated to visualise the performance of each model. When comparing these models, it is evident that the LightGBM and XGBoost models outperform the Artificial Neural Network indicating that the performance of the tree-based algorithms is much better. The results obtained in this research are similar to the work carried out by Pan et al., who also was able to obtain an AUC score of 0.72, similar to the work conducted in this research. The main difference in this research is the tuning of parameters of both LightGBM and XGBoost, from which the results generated to indicate that both the tree-based algorithms were able to achieve the same accuracy. However, the computational time taken for LightGBM to be executed is much quicker than the time taken for the XGBoost algorithm. This is due to the histogram-based splitting, GOSS end exclusive feature bundling, which has also been explained by Ke et al. (Guolin Ke et al., 2017).

Table 10 Comparison Report of Models

Models	Accuracy	Precision	Recall	F1-score	Computational Time Taken
LightGBM	66%	0.66	0.66	0.66	226.4s
XGBoost	66%	0.66	0.66	0.66	3957.1s
Artificial Neural Network	52%	0.54	0.53	0.53	2183.8s

7. Conclusions and Future Work

7.1. Observations

The first observation, keeping the title of the dissertation in mind, is that tree-based machine learning models perform better than artificial neural networks for binary Classification. Both the decision tree algorithms, LightGBM and XGBoost, gave an accuracy 10% higher than the artificial neural network for the same dataset.

From the beginning of this project, it was observed that LightGBM is likely to perform better compared to other machine learning algorithms. Multiple sources from the literature review and projects on Kaggle demonstrated higher accuracy using this machine learning model.

XGBoost, the second tree-based model, gave the same accuracy as LightGBM; however, it took 3957 seconds, approximately 17 times greater than the time taken by LightGBM (226.4 seconds).

It was also observed that Neural Network only gave an accuracy of just over 50%. This algorithm did not perform up to expectations. From the background research and literature review, neural networks had been widely endorsed for Classification – but that was not reflected in this dataset. In retrospect, the dataset features could have been reduced to a smaller number, giving a better accuracy due to small dimensionality.

Considering these observations, we can answer the research questions posed at the start of this report.

Research Question 1: Can machine learning algorithms be used to predict if a windows machine has malware?

Answer: Yes, machine learning algorithms, both tree-based and neural networks, can be used to predict if a windows machine has a malware

Research Question 2: Which machine learning binary classification model performs best in predicting if a windows machine has malware?

Answer: LightGBM is the binary classification model that performs best in predicting if a windows machine has malware.

7.2. Limitations

There were certainly a few limitations of this project. The primary being low accuracy. It is observed that the models could not give an accuracy of over 70%, which is not ideal. This could result from a small number of data samples – as mentioned earlier, the dataset had been shortened from 8 million rows to 1 million due to computation power requirements. Another limitation was time constraints, as seen above, the XGBoost model computational time was over an hour, and hyper tuning parameters took plenty of time.

A step that could have been taken to improve the accuracy would be to certainly have more computation power, which is discussed in the next section, and spend more time treating the missing value in the data. All but one feature's NaN values were filled with the mean of the feature – this step could have been modified into individually exploring the features and deciding the best way to replace the NaN values.

7.3. Future Work

As mentioned before, to improve the performance of the models, they would need to be fed more training data; this would require more computational power, which is why this project could be run on the cloud in the future. For instance, using the Amazon Web Services tool Sagemaker. Amazon Sagemaker helps developers create, train, and deploy machine learning models in the cloud. It is also known to reduce training time exponentially and scale the infrastructure according to computational needs (Amazon Web Services, 2019).

Another feature that could be implemented in the future would be an application with a robust User Interface. This feature would enable the user to enter the properties of their windows machine, like their city if they are a gamer, where the anti-virus products are enabled, their browser identifier, if they have a firewall enabled and so on. Once the input is received manually or automatically collected from the device, the application runs the machine learning model and provides the user with the likelihood of their machine containing malware. This could also be accompanied by a few remediation actions depending on the properties. For instance, the user could be asked to turn their firewall on or update their anti-virus to further protect their devices from malware.

8. Reflections on Learning

This project certainly adheres to the concept of lifelong learning. This one-semester final year project has certainly played a vital role in preparing me for my future career. I decided to select this topic as I have always had a keen interest in cybersecurity. However, I was keen to learn more about machine learning and how this integrates with the field of security. Through the course of completing this research, I have learnt plenty of lessons – for instance, writing a report including various stages in a process requires time management skills. I was certain I would be able to balance this through the help of my supervisor's weekly catchup, however, I found myself falling a little behind during Implementation – this was due to unanticipated issues like time taken to run the models, trial and error for hyper tuning the parameters and so on. This helped me realise that in the future, while planning projects, I should take into account extra time for unexpected circumstances. During the four years at university, this was by far the longest report that required continuous effort and commitment; it had certainly made me confident to take up more significant projects and write reports in my professional life. In the technical aspect, as I aimed, I was able to learn a lot about the domain of machine learning; I now understand the difference between classifiers, parameters and models. I have also understood the importance of data analysis and preparation for any project. After implementing the three models, the evaluation phase helped me gain insight into how to pick the best solution for any given problem and produce a robust solution. Overall, I believe undertaking this project has helped me grow as a computer scientist and helped me practice decision-making, analysis, time management, problem-solving, and research. These skills will help me become a valuable member of the cybersecurity community.

9. References

AbdulNabi, I. and Yaseen, Q. (2021). Spam Email Detection Using Deep Learning Techniques. Procedia Computer Science, 184, pp.853–858. doi:10.1016/j.procs.2021.03.107.

Al-Omari, M., Rawashdeh, M., Qutaishat, F., Alshira'H, M. and Ababneh, N. (2021). An Intelligent Tree-Based Intrusion Detection Model for Cyber Security. Journal of Network and Systems Management, 29(2). doi:10.1007/s10922-021-09591-y.

Amazon Web Services (2019). Build, Train, and Deploy Machine Learning Models | Amazon SageMaker. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/sagemaker/.

Bachman, E. (2018). Which algorithm takes the crown: Light GBM vs XGBOOST? [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/ [Accessed 30 Mar. 2022].

Bensalem, M., Singh, S.K. and Jukan, A. (2019). On Detecting and Preventing Jamming Attacks with Machine Learning in Optical Networks. www.preprints.org. [online] doi:10.20944/preprints201901.0311.v2.

Burnaev, E. and Smolyakov, D. (2016). One-Class SVM with Privileged Information and Its Application to Malware Detection. [online] IEEE Xplore. doi:10.1109/ICDMW.2016.0046.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. [online] Available at: https://arxiv.org/pdf/1603.02754.pdf [Accessed 27 Apr. 2022].

Chigozie, E., Nwankpa, W., Ijomah, A., Gachagan, S. and Marshall (2018). Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. [online] Available at: https://arxiv.org/pdf/1811.03378.pdf.

Climer, S. (2018). Mindsight. [online] Mindsight. Available at: https://gomindsight.com/insights/blog/history-of-cyber-attacks-2018/ [Accessed 10 Feb. 2022].

Fidelis Cybersecurity (2020). Using Machine Learning for Threat Detection. [online] Fidelis Cybersecurity. Available at: https://fidelissecurity.com/threatgeek/network-security/using-machine-learning-for-threat-detection/.

Guolin Ke, Qi Meng, Finley, T., Taifeng Wang, Chen, W., Ma, W., Qiwei Ye and Tie-Yan Liu (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Nips.cc, [online] pp.3146–3154. Available at: https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree [Accessed 8 Nov. 2019].

Helen (2021). 6 Malware Detections/18 Malware Types/20 Malware Removal Tools. [online] MiniTool. Available at: https://www.minitool.com/backup-tips/malware-detection.html [Accessed 15 Apr. 2022].

Holmen, U. (2022). AI and machine learning for threat detection. [online] NTT. Available at: https://services.global.ntt/en-gb/insights/blog/ai-and-machine-learning-for-threat-detection#:~:text=Machine%20learning%20empowers%20automatic%20reasoning [Accessed 3 Mar. 2022].

Jason Brownlee (2016). A Gentle Introduction to XGBoost for Applied Machine Learning. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ [Accessed 30 Mar. 2022].

Jayaswal, V. (2020). Performance Metrics: Confusion matrix, Precision, Recall, and F1 Score. [online] Medium. Available at: https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262 [Accessed 6 May 2022].

Kaggle (2013). Microsoft Malware Classification Challenge (BIG 2015). [online] kaggle.com. Available at: https://www.kaggle.com/c/malware-classification.

Kaggle (2018). Microsoft Malware Prediction. [online] kaggle.com. Available at: https://www.kaggle.com/competitions/microsoft-malware-prediction/data [Accessed 2 Feb. 2022].

Keras (n.d.). Keras documentation: Dropout layer. [online] keras.io. Available at: https://keras.io/api/layers/regularization_layers/dropout/.

Kern, C., Klausch, T. and Kreuter, F. (2019). Tree-based Machine Learning Methods for Survey Research. Survey research methods, [online] 13(1), pp.73–93. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7425836/ [Accessed 15 May 2022].

Kumar, R. and S, G. (2020). Malware classification using XGboost-Gradient Boosted Decision Tree. Advances in Science, Technology and Engineering Systems Journal, 5(5), pp.536–549. doi:10.25046/aj050566.

Leevy, J.L., Hancock, J., Zuech, R. and Khoshgoftaar, T.M. (2021). Detecting cybersecurity attacks across different network features and learners. Journal of Big Data, 8(1). doi:10.1186/s40537-021-00426-w.

Mary Landesman (2021). A Brief History of Malware. [online] Lifewire. Available at: https://www.lifewire.com/brief-history-of-malware-153616 [Accessed 17 Feb. 2022].

McLaughlin, N., Doupé, A., Joon Ahn, G., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E. and Zhao, Z. (2017). Deep Android Malware Detection. Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17. [online] doi:10.1145/3029806.3029823.

Microsoft (2022). Microsoft Defender for Endpoint. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/microsoft-defender-endpoint?view=o365-worldwide.

NCSC (2019). How Cyber Attacks Work. [online] Ncsc.gov.uk. Available at: https://www.ncsc.gov.uk/information/how-cyber-attacks-work [Accessed 17 Feb. 12AD].

Nicholson, C. (n.d.). A Beginner's Guide to Neural Networks and Deep Learning. [online] Pathmind. Available at: https://wiki.pathmind.com/neural-network#:~:text=Neural%20networks%20help%20us%20cluster [Accessed 16 May 2022].

NIST (2019). 1.1.1. What is EDA? [online] Nist.gov. Available at: https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm.

NVIDIA (n.d.). What is XGBoost? [online] NVIDIA Data Science Glossary. Available at: https://www.nvidia.com/en-us/glossary/data-science/xgboost/.

Pan, Q., Tang, W. and Yao, S. (2020). The Application of LightGBM in Microsoft Malware Detection. Journal of Physics: Conference Series, 1684, p.012041. doi:10.1088/1742-6596/1684/1/012041.

phadnispradnya (2021). Streamlined Data Ingestion with Pandas. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/streamlined-data-ingestion-with-pandas/#:~:text=Data%20Ingestion%20with%20Pandas%2C%20is%20the%20process%2C%20of [Accessed 12 Apr. 2022].

Rankin, B. (2018). A Brief History of Malware—Its Evolution and Impact. [online] Lastline. Available at: https://www.lastline.com/blog/history-of-malware-its-evolution-and-impact/ [Accessed 14 Feb. 2022].

Shon, T., Kim, Y., Lee, C. and Moon, J. (2005). A machine learning framework for network anomaly detection using SVM and GA. [online] IEEE Xplore. doi:10.1109/IAW.2005.1495950.

Sidath Asiri (2018). Machine Learning Classifiers. [online] Medium. Available at: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623 [Accessed 2 Mar. 2022].

Tidy, J. (2022). Study: UK firms most likely to pay ransomware hackers. BBC News. [online] 23 Feb. Available at: https://www.bbc.com/news/business-60478725 [Accessed 23 Apr. 2022].

Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T. and Yagi, T. (2016). Malware Detection with Deep Neural Network Using Process Behavior. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). [online] doi:10.1109/compsac.2016.151.

Townsend, C. (2019). Cyber Security Summit New York 2019. [online] United States Cybersecurity Magazine. Available at: https://www.uscybersecurity.net/history/ [Accessed 14 Feb. 2022].