Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Mathias Knutsen
Eivind Hestnes Lervik

Malware detection using supervised machine learning

Bachelor's project in Computer Engineering
Supervisor: Donn Morrison
May 2020

NTNU
Norwegian University of Science and Technology

Mathias Knutsen
Eivind Hestnes Lervik

Malware detection using supervised machine learning

Bachelor's project in Computer Engineering
Supervisor: Donn Morrison
May 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

As more and more services are moving online, the internet has become a lucrative hunting ground for people with malicious intent. Malware is spreading faster than ever, and today's signature-based solutions are not good enough to protect us from every threat. Modern machine learning techniques are proven to be powerful tools against malware. We provide a comprehensive review of published literature on the topic of malware detection using machine learning, covering a number of different approaches. In addition we propose 24 supervised machine learning approaches of our own to statically detect malware. These are based on previous literature for malware detection on Microsoft Windows systems. Our methods can detect unseen malware, referred to as 0-days, with accuracies above 95%. The best performing models include deep neural networks, random forest and more.

Sammendrag

Etter hvert som flere og flere tjenester flytter online, er internettet blitt en lukrativ plass for personer med ondsinnet hensikt. Skadevare sprer seg raskere enn noen gang, og dagens signaturbaserte løsninger er ikke gode nok til å beskytte oss mot enhver trussel. Moderne maskinlæringsteknikker er bevist å være kraftige verktøy for å bekjempe skadevare. Vi gir en omfattende gjennomgang av publisert litteratur som dekker forskjellige metoder for skadevirus gjenkjenning ved hjelp av maskinlæring. I tillegg foreslår vi 24 veiledede maskinlæringsmetoder for å statisk oppdage skadelig programvare. Disse metodene er basert på tidligere litteratur for gjenkjenning av skadelig programvare på Microsoft Windows systemer. Våre metoder kan oppdage ukjent skadelig programvare, kjent som 0-days, med nøyaktigheter over 95%. Et dypt nevralt nettverk samt random forest er blant de beste modellene.

Preface

We chose to research machine learning and malware detection, because we are both passionate about artificial intelligence and machine learning in general. As for malware detection, we are both avid users of the personal computer, and know the amount of threats one encounter on an everyday basis. We wanted to do our own research and gain experience and knowledge on the topic.

We could not have achieved what we have without help. We would like to thank Donn Morrison, our supervisor, for help and guidance. And secondly, NTNU IDI for providing all necessary resources. Thank you for your support.

May 19, 2020, Trondheim

Mathias Knutsen and Eivind Hestnes Lervik

Assignment

Title

Malware detection with machine learning

Purpose

Write a literature review of recent works, determine a good feature set, and compare different machine learning (ML) approaches to detect malicious software.

Description

Malware detection is often signature-based in practice, meaning automatic detection is not possible for 0-day (unseen) malware. Machine learning can help by learning about common attributes of benign software and looking for exceptions, or outliers. Either supervised or unsupervised learning could be applied.

Static and dynamic analysis of software can be combined with a machine learning model for clustering or classification

Vision and requirements document

See vision and requirements document in subsection 9.1 and subsection 9.2 respectively.

Contents

List of Figures

List of Tables

1 Introduction and relevance

1.1 What exactly is malware?

Malware is short for malicious software, which is a category defining any software designed to cause harm or disrupt computer systems [1]. The most common way to spread malware is through email attachments. Any software that causes harm or disruptions while not being designed that way is categorized as a bug and not malware.

Malware comes in all shapes and forms, and we will therefore define a few different well-known types. Defining and understanding the different types of malware is important to figure out how we can detect them. After knowing what distinct features and properties they have, we can classify them using machine learning methods.

The *computer virus* is the type of malware most people are familiar with. A virus is an executable that injects malicious code into other programs and thus replicates itself throughout a system [2]. Microsoft Windows based systems are the absolute most vulnerable.

A *worm* is very similar to a virus as its goal is also to replicate, however a worm's goal is to replicate to other systems [3]. The worms are usually spread around in a network with a security flaw. The biggest difference between a virus and a worm is the fact that the virus aims to harm the computer while the worm slows or harms the network.

The *trojan* got its name from the ancient greek story of the deceptive trojan horse [4]. A trojan is a malware that disguises itself and hides its true intent. Once executing the trojan it will infect your system, however it does not replicate like a virus or worm. Trojans are usually used to steal personal information from the user.

Ransomware is relatively new, and as its name implies, a ransomware malware will demand a ransom after encrypting personal files. The ransom is usually paid in Bitcoin or with other cryptocurrencies. The type of malware is typically carried within a trojan tricking a user to executing it [5].

1.2 History of influential malware outbreaks

Five years before the first malware was created, John Von Neumman published "Theory of self-reproducing automata", where he describes a self-reproducing malware [6]. Little did he know that in 1971 someone would pick up his idea and create the world's first malware [7]. Bob Thomas from BBN Technologies made a malware testing Neumman's theory. He named the malware *Creeper* as it displayed "I'm the creeper, catch me if you can!". Creeper spread through the ARPANET until it was terminated by Reaper, a program made to eradicate Creeper [8].

In the 1980s two major outbreaks happened. The first one in 1981, Elk Cloner, a malware targeting Apple II. The latter took place in 1988, The Morris

Worm. This was the first computer worm and gained a lot of media attention as the damages are estimated to be between $100,000 and $10,000,000 [9].

In 2000, the "ILOVEYOU" worm emerged. Within a few hours of its release it had already infected millions of computers worldwide causing billions of dollars in damage [10]. The worm was spread through email attachments and was titled "ILOVEYOU" hence the name. Opening the attachment launched a Visual Basic script overwriting files and forwarding the email to all contacts.

Seven years later in 2007 the ZeuS trojan was discovered. The trojan was designed for cyber theft and ended up stealing more than $70 million in total. The FBI arrested over 100 hackers related to the ZeuS trojan [11]. In 2009 another group of hackers targeted tech companies like Google in an attempt to install malware in their source code.

The Stuxnet worm was a malware that is today known as a cyberweapon. Stuxnet is believed to be developed as far back as 2005 with the purpose to cause damage to nuclear programs. It is believed that the United States and Israel worked together to make the worm and caused substantial damage to Iran's nuclear systems [12]. The worm was first detected by VirusBlokAda in 2010 [13].

CryptoLocker and WannaCry took place in 2013 and 2017 respectively. These are the biggest ransomware attacks that have taken place as of 2020. CryptoLocker was spread through emails and encrypted user's hard drive, asking for money to decrypt the files [14]. WannaCry was a lot like CryptoLocker but got eradicated a few days after discovery. The damages were still close to $4 billion [15].

1.3 Machine learning and the battle against malware

Malware is more common than ever, especially on Windows systems. The damages can be extensive depending on what is targeted, networks or personal computers. To combat malware, we use anti-malware software and firewalls, but do these options really protect us from every potential threat? Traditional anti-malware products are mostly signature-based, which means that they use a file's signature to identify if it's a known threat or not. This means that so-called zero days will not be discovered by the anti-malware and might infect the system or network causing substantial damages.

Traditional malware detection can be divided into four different categories; pattern matching, heuristic analysis, behavioral analysis and hash matching [16]. Pattern matching is when you take a few specific bytes from a file and make a signature based on this. For future scans you will match your program to a signature database to check if it is a known malware or not. Heuristic analysis is a little different, yet a signature-based approach. The anti-malware scans a file checking for methods and code common for malware. Then makes a signature based on these heuristics. Behavioral analysis is when an anti-malware executes a program to check what it does on a system level. After execution it concludes if what was just run was a malware or not, maybe not the best idea

for a personal computer. Hash matching is when the anti-malware calculates hashes from different parts of the file and then saves it to a signature database.

All the methods mentioned above can be easily bypassed by taking the same malware, changing a few bits throughout the file, and using other heuristics. To solve this flaw, we can use machine learning. Machine learning can learn what properties a malware has and identify new types without relying on a signature database. With machine learning we can look at features extracted from a malware executable, identify it and block it before traditional anti-malware lets it through the firewall.

1.4 Research question and report structure

Having introduced what malware is, and the need for machine learning on the topic, we introduce our research question. *How does different supervised machine learning approaches to static malware detection compare?* We will continue to research this question, starting off with introducing different machine learning methods in general, before looking at existing literature on the topic. We will then propose different approaches of our own, discuss the results, and reach a conclusion on the matter.

1.5 Acronyms and abbreviations

See subsection 9.5 for a list of acronyms and abbreviations.

2 Theory

2.1 The data

Before any machine learning can be done, we need data for our algorithm to train on. This data is usually called a training dataset and contains multiple *features* x for each entry and often a correct label y. A feature is a variable that says something about the entry. For a car dataset the wheel size and car color would be good features. The label could be something like a small car, a big car, or even brands. The machine learning algorithm will try to see correlations and patterns between the features to predict the label for new unknown entries.

2.2 Preprocessing

After acquiring a dataset, it is important to verify that the data is good and ready for a ML algorithm. Data with features of varying magnitudes might affect the algorithm in an unwanted way. To counter this, one can normalize the data by scaling the features. An example can be a dataset with file size and extension as features. As the file size probably has a higher magnitude than file extension, the ML algorithm might prioritize the enormous file size. After scaling the data, the algorithm will weigh them both equally [17].

Another thing to note is the balance of the dataset. Let us say we have a dataset with malware and benign software. If the amount of malware samples is higher than the number of benign samples, the algorithm might once again weigh the malware higher than the benign and achieve unwanted results. To counter this, one should try to balance the dataset by either generating artificial benign entries, or reduce the number of malicious entries in the dataset.

2.3 Supervised vs unsupervised learning

Before we start diving into what algorithms and models that exist, it is important to know the difference between supervised and unsupervised learning. When performing supervised learning you will feed a machine learning algorithm features and the correct label during training, and therefore supervise the algorithm. On the other hand, unsupervised learning does not need any label to train. Unsupervised algorithms categorize data based on the features. This is great for clustering data without labels, and for visualizing correlations between the entries in the dataset. Unsupervised learning can achieve as good results as supervised, and neither method is better than the other. They both have their own use cases.

2.4 Regression and classification

Regression is a much-used supervised machine learning method. Regression methods tries to estimate a function f with features x to a label or continuous value y. Regression is usually used with simpler data with fewer features. The

regression can be linear or polynomial depending on what fits best with the given data. The most known regression method is linear regression.

Classification is a supervised machine learning method that tries to identify what class or category an entry in the dataset belongs to. Classification methods try to estimate a function f with features x to a categorical output y. Classification is a popular machine learning method and there are many different classification algorithms.

2.4.1 Decision tree

A decision tree (DT) is a tree-based method capable of both classification and regression. There are many tree-based methods, but we will describe the most common one, CART. The tree starts off at its root and splits the dataset into two sub parts. A feature and its split point are created based on an information gain metric. The two new nodes keep splitting and creating split points until a max depth is reached. When a max depth or node count is reached, the leaf node will output the mean of the feature labels remaining in the sub part. An example of a decision tree and its 3D plot can be seen in Figure 1 [18].

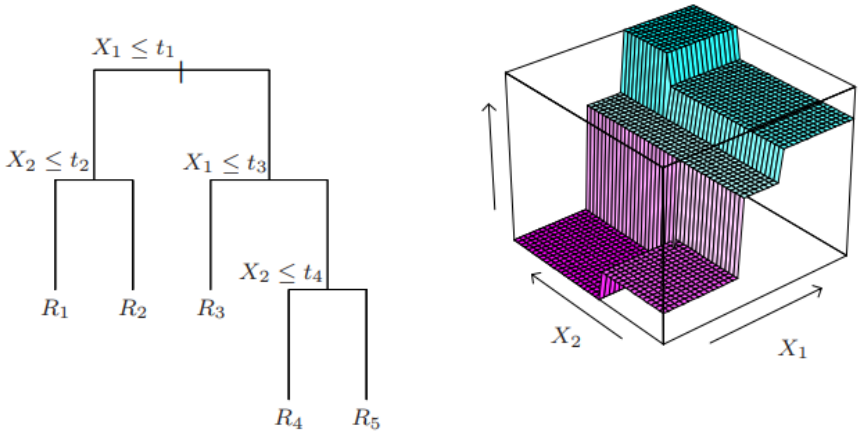


Figure 1: Decision tree and 3D plot visualization on random data [18]

2.4.2 Random forest

Random forest (RF) is another classification method that takes advantage of Decision Trees. RF works by first creating a subset of the original data with N samples. It then grows a decision tree with m random features and min node count n. This process continues until it has grown B trees. To predict new samples, it uses a majority vote for classification, and an average for regression [18].

2.4.3 Gradient boosted trees

Gradient boosted trees (GBT) are other competitors to RF. Instead of generating B number of independent random trees, GBT makes one and one tree in a sequence. In this sequence of trees, the next tree will correct the error of the previous, and therefore "boost" its accuracy. Each tree in a GBT will usually be a "weak learner". This is a simple decision tree with few nodes and low complexity. Because of its sequential nature and dependency of the previous tree, GBT can be time consuming and often impossible to run in parallel like RF [18].

2.4.4 Support vector machines

Support vector machines (SVM) is a set of methods used both for regression and classification. SVM is based on drawing lines between data points, where one side is one class, and the other another class. SVM will try to maximize the line width to classify the data with high confidence. The lines depend on the kernel used and can be both linear and polynomial [18]. An example of a SVM model with a linear kernel can be seen in Figure 2.

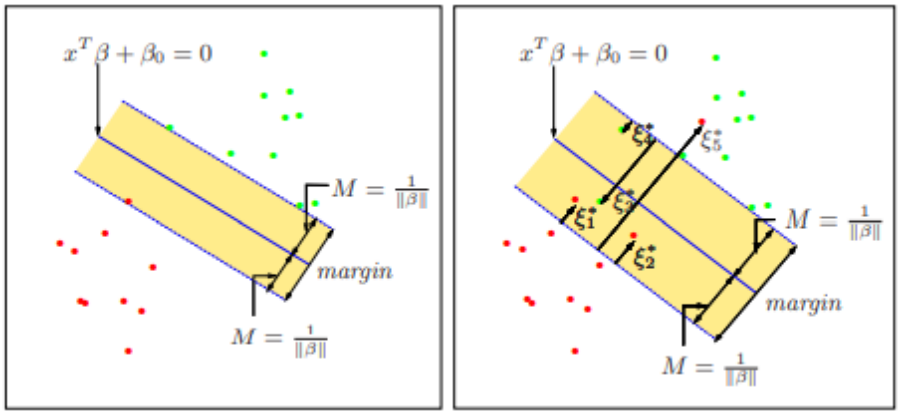


Figure 2: Two simple linear SVC examples [18]

2.4.5 Naive Bayes

Naive Bayes (NB) is a method used for classification. NB works well on datasets with a large number of features. NB assumes that all features are independent of each other. This is not necessarily the case, but it is the idea behind the Bayes' Theorem. NB trains on data by checking each feature one by one and setting up the probabilities of the feature being true and false. This is done for every feature in the set. When predicting a new sample, the probability of being true is combined with all the other true probabilities, and the same for the false ones. The probability with highest confidence wins [18].

2.4.6 K-nearest neighbor

The k-nearest neighbor (kNN) algorithm is a supervised machine learning algorithm that can be used for both regression and classification. It works by looking at the k number of data points that are closest to the input data. When used as a classifier it tries to assign a class by looking at the label of nearby data points and taking the mode of the labels. When used for regression, it instead calculates the mean of the labels of the nearest data points. It works on the assumption that data points that are close together are similar to each other. It can also be useful to assign weights, so that neighbors that are nearer contribute more. One downside to the kNN classifier is its scalability, as it contains all the dataset entries. [18].

2.5 Neural networks

2.5.1 Artificial neural networks

Artificial neural networks (ANN) are a very popular group of supervised machine learning systems that often outperforms other ML algorithms. The idea behind an artificial neural network is to mimic the behavior of biological networks of neurons. A neuron can send and receive a signal from other neurons it is connected to. Whereas biological neural networks work with biochemical processes and electric impulses, artificial neural networks deals with numbers. The neurons in an ANN are structured in layers, with the idea being that the different layers are able to "learn" different distinctive patterns or features. Neurons in one layer are connected to neurons in the next layer. The input layer takes a vector containing the vectorized input features. The last layer of neurons is the output layer and contains the output values. The layers between the input and output layers are called hidden layers and ANNs with multiple hidden layers are called Deep Neural Networks (DNN). The basic layers, where each neuron receives an input from every neuron in the previous layer and gives an output to every neuron in the next layer, are called fully connected layers, or dense layers.

Each neuron in the network can hold a number and has its own bias number. Each connection between two neurons has a weight. The value of a specific neuron is calculated by adding together the value from each neuron connected to it in from the previous layer multiplied by each of the respective connection weights. Then the bias number is added before the result is fed into an activation function. See Figure 3 for illustration. An example of a common activation function is the sigmoid function that takes a number as input and squishes it to a value between 0 and 1. Sometimes a threshold is used, so that the value in a neuron is only transmitted if it is higher than the threshold. This process is done for every neuron in the model. The network's "guess" is given by the values of the neurons in the output layer. When the network is "training", it is the various weights and biases that are adjusted so that the model classifies the training input correctly according to its label. This is done using an algorithm

called backpropagation. In essence, when training a network, you are creating a complex function that when given some input, gives the correct output [19].

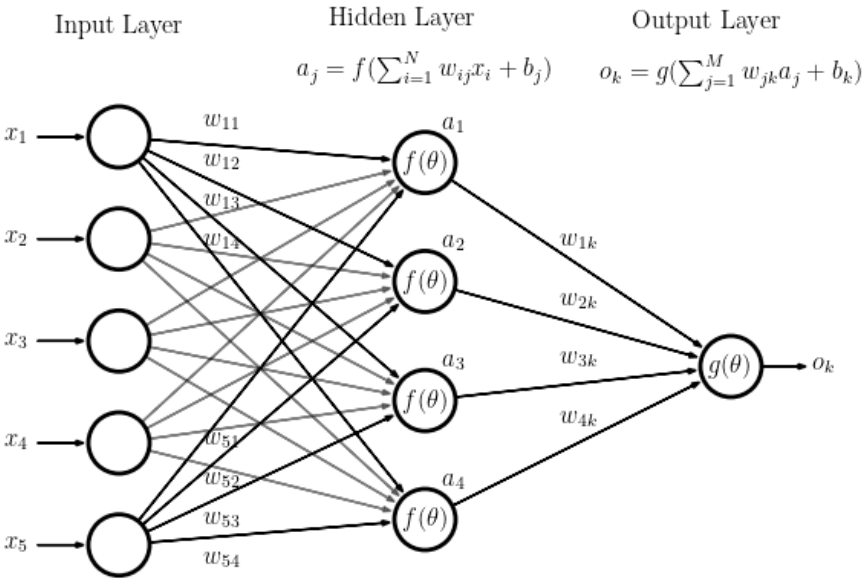


Figure 3: Visualizing a neural network [20]

2.5.2 Convolutional neural network (CNN)

A convolutional neural network is a popular variant of neural networks. CNNs are often used in image classification as they are remarkably good at determining distinct features within an image. The name comes from a specific type of hidden layer called convolutional layer. What happens in these layers is that one or more filters (sometimes called kernels) slides or convolves, across the input matrix (image). The filter is essentially just a small matrix that contain certain values. When it slides across the input, it calculates the dot product of the filter and the particular place of the input it is covering. When it has traversed over the whole input image, the output is now a new "image" known as a feature map, that hopefully highlight specific features about the image. The first convolutional layers can maybe detect general features like vertical and horizontal edges, but with more layers, smaller and more specific features are found. Between each convolution, one usually adds a pooling layer. This layer simply reduces the size of the matrix by either using the highest values in each subsection or using averages. Figure 4 visualizes how a CNN works. The output from the convolutional layers can then be fed into a dense layer and be processed like in a regular NN [21].

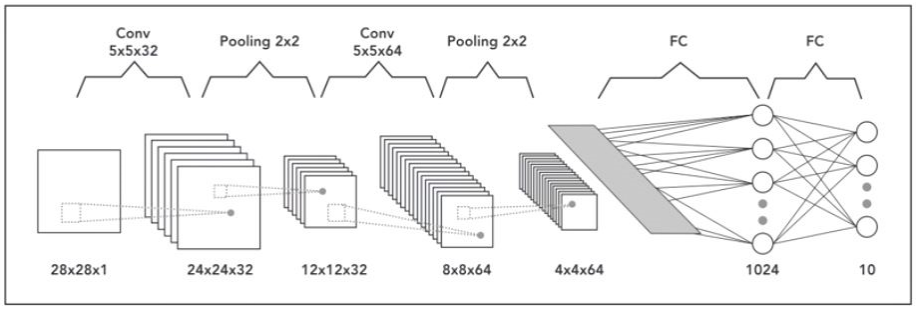


Figure 4: A visualization of the CNN structure [22]

2.5.3 Recurrent neural network (RNN)

Recurrent neural networks (RNN) is another variant of neural networks. They are often used in natural language processing as they focus on features over a time frame (sequential information). The RNN architecture consists of multiple layers, one for each time frame, and has the ability to look back in the previous layers to understand context. The most commonly used RNN is Long short-term memory (LSTM), which functions in the same way but has optimizations to keep a better memory [23].

2.6 Feature selection

Training ML models can be time consuming, and this is often because of redundant features in the dataset. Training on data that is uncorrelated to the label is inefficient. Feature selection are different methods for reducing the dimensionality of the dataset and therefore increasing performance and sometimes even accuracy as redundant features are taken out of consideration. Common feature selection methods include removing features with low or no variance, using statistical tests to choose only the best k features and extracting features from already established ML models [17].

2.7 Evaluating a machine learning model

A machine learning model might be able to make predictions, but are they correct? How does one score a model? To understand this, we need to take a look at model metrics and scoring. These metrics are often given in percent or by a value between 0 and 1. To understand the metrics we are going to cover, it is important to understand the difference between the different predictions in Table 1. By plotting these four values into a matrix, one gets a confusion matrix. This matrix indicates how "confused" a given model is by showing what values are predicted.

True Positive (TP)	Correctly predicted positive
False Positive (FP)	Incorrectly predicted positive
True Negative (TN)	Correctly predicted negative
False Negative (FN)	Incorrectly predicted negative

Table 1: Possible predictions

2.7.1 Accuracy

The most common scoring method of a ML model is accuracy score. The accuracy is determined by $\frac{TP+FP}{N}$ where N is the total number of predictions. As one can see accuracy is a simple scoring that tells you how many correct predictions the model did. This scoring does not say anything about what was predicted wrong.

2.7.2 Precision, recall and F1

Precision, recall and F1 are three other popular scoring methods. Precision is given by $\frac{TP}{TP+FP}$, while recall is given by $\frac{TP}{TP+FN}$. In other words, precision says something about how many true samples compared to false negative samples one predicted, while recall tells you how many true samples the model was able to predict. Recall is often referred to as True Positive Rate (TPR) or detection rate (in a malware context). F1 is a combination of precision and recall aiming to give an overall scoring. F1 is given by $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$.

2.7.3 Receiver operating characteristic

The receiver operating characteristic (ROC) curve is another metric often used to document classification models. The ROC curve is a graph that plots True Positive Rate against False Positive Rate (false alarm rate). One often plots multiple ROC curves in the same graph to compare different models.

3 Literature review

3.1 Introduction

In this section we will look at some of the published work in malware detection and analysis using machine learning. Our primary focus in this paper will be on malware detection on Windows Portable Executable (PE) files using static analysis, but in this section, we will also cover published works on dynamic analysis. Some of the works covered use similar approaches to us and can therefore be more easily compared. Some of the works use datasets that they have gathered themselves or that are no longer available, making their results harder to gauge. When choosing literature to review we wanted articles containing modern and thorough research. We read a lot of various articles from credible sources and chose the most relevant and interesting ones to review. Although our focus is the detection of malware, not classification, some of the works covered will be on the classification problem. This is due to the fact that these problems are closely related, and the approaches used for the two problems are often very similar.

3.2 Malware detection using static analysis

Analyzing a program without executing it is called static analysis. There are many different features that can be extracted from a file without executing it (see chapter 4 for some of the features used in our static dataset). These features can be used in conjunction with a machine learning model to detect whether a file is malicious not. Static data has the advantage that it can be collected quickly, but a classifier based on static data might struggle with obfuscated, encrypted or completely new malware.

Several popular machine learning approaches are compared by Vinayakumar et al. (2019) [24]. Training the models on a dataset of features from 42 140 benign and 41 860 malicious files from the public EMBER dataset [25], they claim impressive results of well over 90% accuracy using kNN, RF, SVM and DNN classifiers. The best being 98.9 with a tuned DNN model. Logistic Regression (LR), Naive Bayes (NB) and AdaBoost (AB) models perform considerably worse with LR and NB models getting classifying accuracies of just 54.9% and 53.8% respectively.

Belaoued and Mazouzi (2015) [26] suggest a feature selection technique based on the Chi-square test [27] to create a method for classifying malware fast using a RF model. They claim to be able to classify a file in just 0.077 seconds with an accuracy of 97.25% on their test set, though it should be noted that the dataset used is rather small at just 338 malicious and 214 benign files. Saxe and Berlin (2015) [28] train a deep neural network with 2 hidden layers on a large dataset of over 400 000 executables. They claim a true positive rate (TPR) of 95.2% with a false positive rate of 0.1% on their test set. However, when training their model on data from before a certain date, and testing it on data from after that date, the TPR dropped to just 67.7%, highlighting that models trained on

static data might struggle with detecting brand new malware.

Another approach to analyzing malware is to convert the malware binary file into an image, which can then be analyzed using computer vision techniques before being classified using machine learning. A technique for visualizing malware binary files into gray-scale images was proposed by L.Nataraj et al. (2011) [29]. To visualize a malware file, the binary is read as 8-bit integers, where the value 0 represents a black pixel and a value of 256 represents a white pixel. The width of the image is fixed while the height varies with the file size. An example of a malware image can be seen in Figure 5.

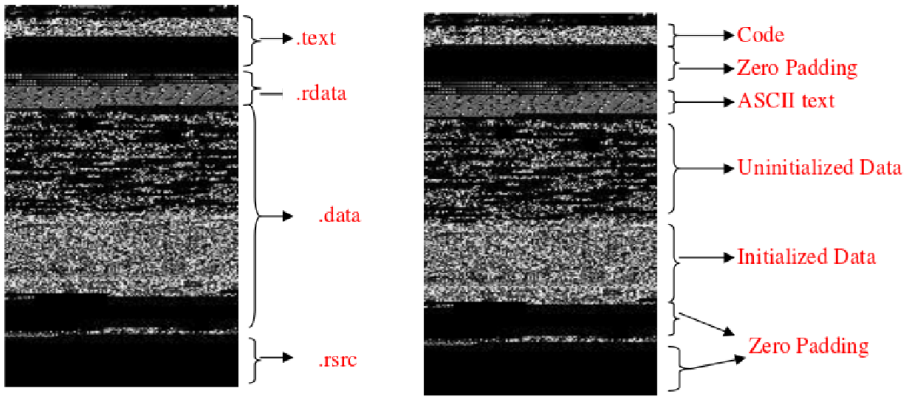


Figure 5: Visualizing malware sections in PE files [30]

According to the paper, images generated from malware of the same family appear very similar to each other, and different from the images of malware from different families. The paper proposes using the generated images to classify malware using image classification techniques. Using a k-nearest neighbor (kNN) classifier using 10 fold cross validation and looking at the 3 nearest neighbors ($k = 3$), on a dataset consisting of 1713 malware images belonging to 8 different families, they manage to achieve a classification rate of 0.9993 average over 10 tests. They also suggest that the images from executables of the same family will look similar after packaging and claim an overall classification accuracy of 0.9808 on data where the packed malware of a specific family is treated as a new family. Thus, suggesting that this approach is quite resilient to obfuscation.

Based on the technique for generating gray scale images from malware binaries presented in the aforementioned paper, several approaches for analyzing malware images are proposed by Luo and Lo [31]. The paper proposes the use of a Local Binary Patterns (LBP) [32] visual descriptor on a binary malware image. The LBP descriptor is used for texture analysis and classification. A LBP feature vector is created by comparing a pixel in the image to its 8 neighboring pixels and assigning a value of either 1 or 0 depending on if its value is greater or less than that of the center pixel. Each value is then multiplied by a

weight and then all are added together to get the new value of that pixel. This is repeated for every pixel in the image. The final descriptor is obtained by calculating the histogram. This descriptor is then fed into a machine learning algorithm for classification.

The dataset they used consists of 12 000 malware images, where 20% of each malware family is used for training the model and the rest for testing. To classify the data, several different approaches are proposed. Support Vector Machines (SVM) and k-Nearest Neighbor classifiers yield accuracies of 87.88% and 85.93% respectively. The best result is achieved using a CNN model for classifying. The network is a simple CNN with two convolutional layers. Using this network, a claimed accuracy of 93.17% is achieved. On a similar dataset of 9 339 malware images, Vinayakumar et al. (2019) [24] claims an accuracy of 96.3% classifying malware into 25 families using a CNN model with a LSTM layer.

3.3 Malware detection using dynamic analysis

Dynamic or behavioral analysis is done by letting the executable that is being analyzed run on a system in order to observe how it behaves. When using dynamic analysis to detect malware, the file is allowed to run in a sandbox environment, to not cause any real damage to the system. While the file runs, information about its behavior, such as system calls or resource usage, is collected and can be used as features for a machine learning system. Based on the assumption that malware must enact certain behaviors to achieve their goal, dynamic analysis has the potential to be very effective in detecting malware. As such it is potentially also very resilient against obfuscated malware. However, modern malware can employ various techniques to check whether it is running in a virtual environment in order to avoid detection [33]. Additionally, the use of a sandbox will potentially also require more resources to run.

Creating a dataset suitable for machine learning with dynamic analysis requires running the files and collecting data about their behavior. Firdausi et al. (2010) [34] compares several different machine learning methods trained on dynamically acquired data. Their dataset consisted of features, primarily API calls made during execution, from 220 malicious and 250 benign files. Several different models are proposed, but the best result is an accuracy of 96.8% using a J48-decision tree model and no feature selection. It should be noted that all 250 benign files were collected from the System32 folder in Windows XP, which may not be a good representation of benign software encountered elsewhere.

The use of API calls to the operating system as behavioral features for dynamic analysis is quite logical. However, Rosenberg (2017) [35] suggests that dynamic malware classifiers based on API calls can be manipulated to falsely classify malicious files as benign. This is shown by modifying malicious code, without removing the malicious functionality. Rhode et al. (2018) [36] suggests a classifier and dataset based purely on machine activity during runtime, rather than API calls. The dataset looks at 10 different features from the runtime of 2345 benign and 2286 malicious executables. The 10 features collected are

CPU usage of the system, user CPU usage, number of packets sent and received, bytes sent and received, memory usage, swap usage, total number of processes running, and the maximum process ID assigned. Using an RNN model trained on the dataset for classification, an accuracy of 96.01% is achieved on a test set of recent malware after 20 seconds of execution. Using the same dataset, Vinayakumar et al. (2019) [24] claims an accuracy of 96.6% using a CNN model.

3.4 Conclusion

The literature shows that a number of various machine learning approaches can be effective in the battle against malware. Both static and dynamic analysis yield very high accuracies in the reviewed literature, although there are some suggestions that static approaches can struggle on newer types of malware. Certain difficulties for dynamic methods was also highlighted by the literature. Various forms of neural networks seem to perform very well across the board. Classifiers like SVM, kNN and decision tree-based models also get good results.

Based on various articles, converting a malware binary into a gray scale image appears to be a very solid approach to malware classification using static analysis with some clear advantages. It is easy to process, appears to be resilient against obfuscation and it can take advantage of image analysis techniques as well as CNNs remarkable ability to process images. While several different articles obtain very good results when classifying malware into various families, we would have liked to see how such a technique performs for malware detection.

4 Choice of technology and approach

4.1 The data

While malware is found on many different operating systems, Windows is the most targeted non-mobile system, due to the fact that it has by far the most users. As a result, there exists a huge amount of malware for the Windows operating system. This means that there is a lot of data available, as well as a need for reliable and accurate malware detection methods. Executable files on the Windows system use the Windows Portable Executable (PE) file format and this is the format we will be working with in this thesis.

4.2 The PE format

The PE format was created by Microsoft along with the release of Windows NT 3.1. It is a file format for executables on Windows systems that has the information needed for Windows to execute the wrapped system code. The format consists of multiple headers and regions used by the operating system to map the file into the memory, so that it can execute correctly using the required shared libraries (See Figure 6). Below we will look at some of the important parts of the PE format. Detailed information about all the fields can be found in the Microsoft documentation [37]. These headers give detailed information about the file and can therefore be used as features when designing a machine learning model for detecting malware.

The Common Object File Format (COFF) header is the first part of the PE format, and contains various important information about the file. What kind of machine the program is intended for (Intel, ARM etc.), file type (dll, exe, obj), number of sections and symbols are some of the information found in the COFF header. In other words, the COFF header contains important information and system requirements.

The next part is the optional header and is only required for executable files (called optional because it is not required for object files). It provides useful information for the operating system loader responsible for executing the file. The size of the optional header is not fixed and is specified in a field in the COFF header. The data directories in the optional header gives sizes and addresses of strings or tables used by Windows, for example tables for imports, exports, resources, and exceptions.

The section table follows immediately after the optional header. It consists of a 40-byte header for each of the file's different sections. Sections are the actual content of the executable file and contains code, data, resources, and other information.

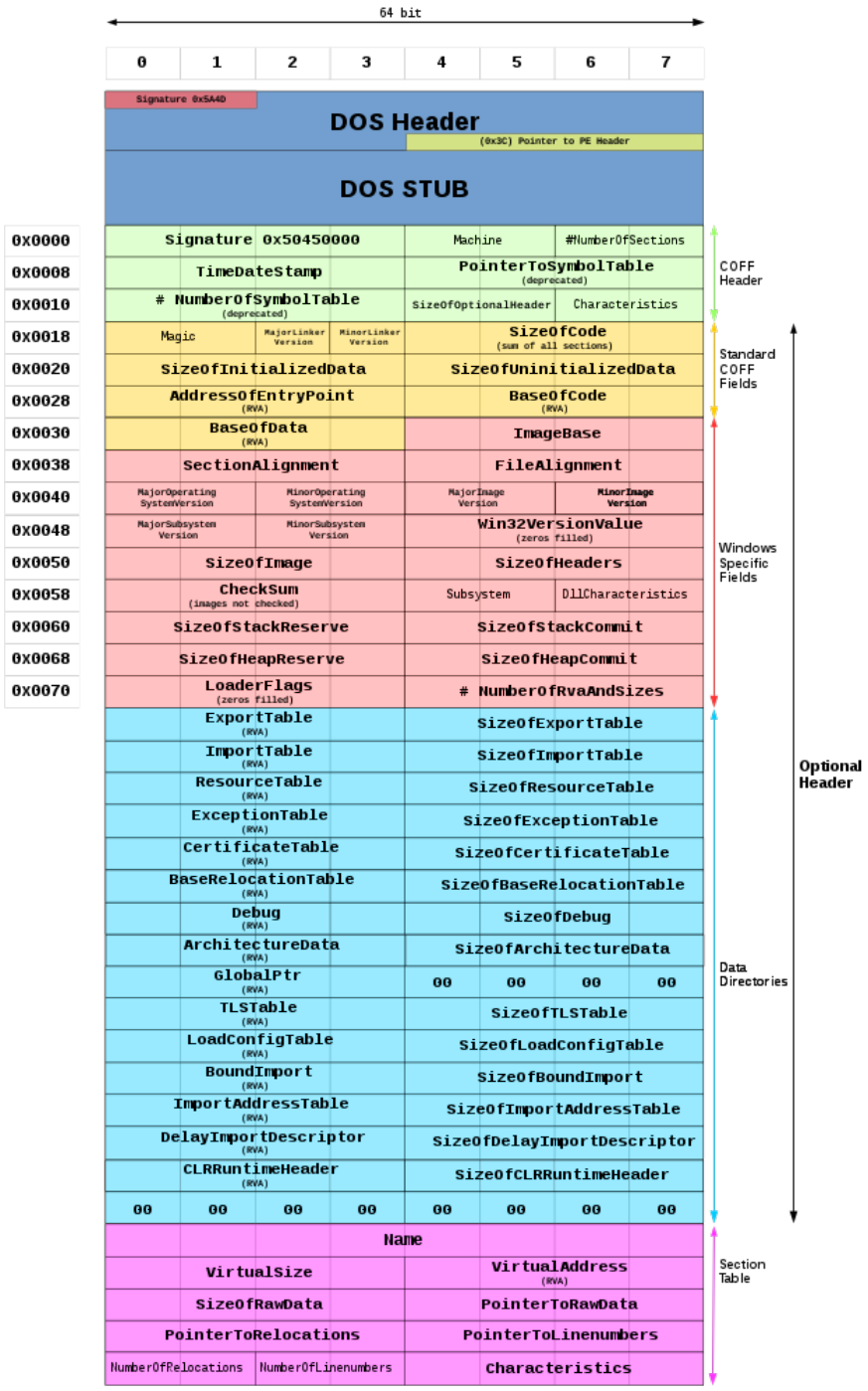


Figure 6: PE file structure [38]

4.3 Dataset

When using ML for malware detection, the malware file itself is not fed directly into the model. Instead, the file must be analyzed in some way and useful features must be extracted. When deciding on a dataset for our models, we wanted a large dataset with a wide range of different features so that it provides good insight into how the file works. We also decided on using a dataset based on statically obtained features as it makes the models faster and more accessible since it does not require the use of a virtual environment. Since new malware evolves over time in order to avoid available detection methods, a malware dataset for detecting new malware should contain modern samples.

Based on these criteria, we chose the EMBER dataset created by Endgame [25]. It is a modern dataset from 2018 and contains features extracted from 1.1 million Windows portable executable (PE) files and has a total of 900K training samples, where 300K are malicious, 300K are benign and the last 300K are unlabeled. The dataset is fully balanced which is of great importance when doing supervised learning. In addition to the training data the dataset also includes 200K test entries. We will only use the labeled entries, as we are doing supervised learning. Bundled with the dataset is a LightGBM model, a well-known gradient boosted tree model. We will not include this model in our thesis as it has been extensively tweaked and is not comparable to our proposed models.

The EMBER dataset consists of a collection of JSON entries, each describing a malicious or benign PE file. Each of these entries contains the following data:

- SHA-256 hash of the original file
- A timestamp indicating when it was first detected
- A label; 0 for benign, 1 for malicious and -1 for unlabeled (unlabeled data can be used for unsupervised learning and is therefore included)
- Features extracted from parsing the PE-file
- Format-agnostic histogram features

The PE files are parsed using the Library to Instrument Executable Format (LIEF) [39]. This extracts useful information about the file from the headers and sections described in subsection 4.2. Features received from parsing includes general file information, header information, list of imported and exported functions and information about the various sections. Three different format-agnostic (i.e. these features are extracted without parsing the file) feature groups are also included: Byte histogram, byte-entropy histogram and string information.

The byte histogram gives the count of each byte in the file. A byte (8 bits) can have a value ranging from 0 to 255. The histogram is a vector of size 256 where each index represent the corresponding byte value and the value at a specific index is the number of times that specific byte appears in the file.

```
"sha256": "000185977be72c8b007ac347b73ceb1ba3e5e4dae4fe98d4f2ea92250f7f580e",
"appeared": "2017-01",
"label": -1,
"general": {
  "file_size": 33334,
  "vsize": 45056,
  "has_debug": 0,
  "exports": 0,
  "imports": 41,
  "has_relocations": 1,
  "has_resources": 0,
  "has_signature": 0,
  "has_tls": 0,
  "symbols": 0
},
"header": {
  "coff": {
    "timestamp": 1365446976,
    "machine": "I386",
    "characteristics": [ "LARGE_ADDRESS_AWARE", ..., "EXECUTABLE_IMAGE" ]
  },
  "optional": {
    "subsystem": "WINDOWS_CUI",
    "dll_characteristics": [ "DYNAMIC_BASE", ..., "TERMINAL_SERVER_AWARE" ],
    "magic": "PE32",
    "major_image_version": 1,
    "minor_image_version": 2,
    "major_linker_version": 11,
    "minor_linker_version": 0,
    "major_operating_system_version": 6,
    "minor_operating_system_version": 0,
    "major_subsystem_version": 6,
    "minor_subsystem_version": 0,
    "sizeof_code": 3584,
    "sizeof_headers": 1024,
    "sizeof_heap_commit": 4096
  }
},
"imports": {
  "KERNEL32.dll": [ "GetTickCount" ],
  ...
},
"exports": []
"section": {
  "entry": ".text",
  "sections": [
    {
      "name": ".text",
      "size": 3584,
      "entropy": 6.368472139761825,
      "vsize": 3270,
      "props": [ "CNT_CODE", "MEM_EXECUTE", "MEM_READ"]
    },
    ...
  ]
},
"histogram": [ 3818, 155, ..., 377 ],
"byteentropy": [0, 0, ... 2943 ],
"strings": {
  "numstrings": 170,
  "avlength": 8.170588235294117,
  "printabledist": [ 15, ... 6 ],
  "printables": 1389,
  "entropy": 6.259255409240723,
  "paths": 0,
  "urls": 0,
  "registry": 0,
  "MZ": 1
},
```


Figure 7: Visualization of raw features [25]

The byte-entropy histogram is included to say something about the byte entropy in different sections of the file. The byte-entropy histogram is calculated by dividing the file into equal sections and calculating the entropy for each one. Before transforming the raw features to trainable data, the byte-entropy histogram is normalized to a fixed length.

The string information includes various statistics about the strings in the file. Character strings are defined as strings of at least 5 printable characters. The number of strings, average length, a histogram of the characters and the entropy of characters across the strings are used as features in the dataset. The occurrence of certain string patterns, like C:\ indicating a path, are also tracked. The raw strings themselves are not included for privacy reasons.

With the parsed features as well as the format-agnostic features, the dataset contains a total of 2351 different features for each entry after vectorization. A visualization of some of the raw features extracted from a PE file can be seen in Figure 7.

4.4 Proposed models

As our research question suggests, we are going to compare different supervised machine learning approaches to statically detect malware. We are comparing a total of 24 ML approaches. We have used different techniques for feature selection to compose four different feature sets that we use to train six different models. We have chosen DNN, RF, GBT, DT, SVM and NB. The following models are by far some of the most popular ML algorithms to date. Other popular algorithms like kNN have not been included because of the sheer size of our dataset. Deep Neural Networks have shown to be particularly good at detecting malware judging by previous literature. Random forest and gradient boosted trees are both based on decision trees. Decision trees are often compared against neural nets, and we therefore included these methods. SVM has proven to perform good with small datasets and polynomial kernels previously, but we are curious about using a linear kernel with a larger dataset. Naive Bayes is a very basic algorithm and have proven to give lacking results in the past, but we want to find out if this is true. We will only be focusing on supervised learning, and the unlabeled entries from the EMBER dataset will therefore be omitted. All preprocessing and machine learning have been done on a virtual machine with 32 cores and 32GB of memory.

4.4.1 Normalization

Using EMBERs built in function, we can load an already vectorized dataset with only numeric values. The large variance in feature magnitudes still poses a problem for some machine learning methods like neural networks. To counter this, we use a scaler to normalize all the features in the dataset. The scaler is calculating new values based on a statistical z-score [17].

4.4.2 Feature selection

Before feeding the dataset to our models, we generated four different feature sets using popular feature selection methods. We chose to perform feature selection to increase training speeds, and also reduce the size of our models. We could have chosen one method and stuck with it, however, comparing multiple methods can give useful insight in general feature selection and what works best. The four different selection methods used are available in the scikit-learn library [17].

The first method is removing all features with no variance. This means that all features with the same value for all entries will be removed. No classifier will benefit from a feature with no variance. After this selection, we end up with 2335 features. The second method chooses the top 512 scoring features based on the chi-squared test [27]. The third method is recursive and uses a tree classifier to test at what number of features the accuracy is at its highest. As seen in Figure 8 the accuracy is relatively similar at all steps except 0 and 1. The optimal number of features is right around the small bump at step 14 with 447 features. This method is time consuming as it creates a tree classifier for each step. The fourth and final method is extracting the features used by an already trained tree classifier. This classifier used 675 distinct features. Using a small subset of the original dataset will be faster and might produce better results. From here on, we will refer to the feature sets as feature set 1, 2, 3 and 4 respectively.

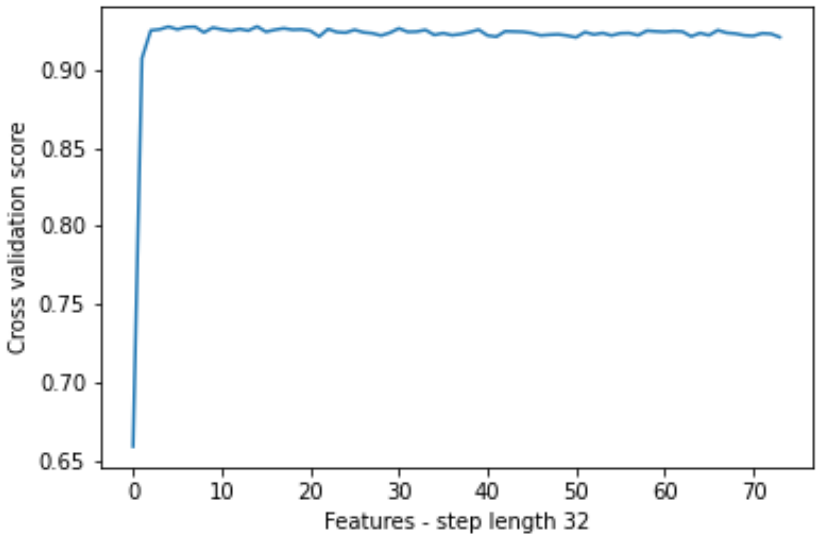


Figure 8: Recursive feature selection with cross validation scores

4.4.3 Scikit-learn

Scikit-learn is an open-source ML library that implements multiple algorithms and tools for classification, regression, clustering, and data processing [17]. We used scikit-learn for all our classification models. These models include RF, GBT, DT, SVM and NB. Minimal parameter-tuning has been done so that we can easily compare the five algorithms. It is worth mentioning that the SVM is running a linear kernel as a polynomial kernel would be too time consuming due to the dataset size. The GBT model we will be using is an experimental model by scikit-learn named histogram based gradient boosting (HGB). All the scikit-learn models are strict classifiers and return either zero or one. One means malware, and zero is benign. See subsection 2.4 for more details on how the algorithms work. Scikit-learn also has implementations of neural networks, however we will be using Tensorflow.

4.4.4 Tensorflow

Tensorflow is an open-source platform for deploying different machine learning models. Tensorflow offers different level APIs including the high level API, Keras, which we used to create our deep neural network. We are using Keras as it easily blends with the scikit-learn workflow [40]. The previously mentioned scikit-learn classifiers are mere algorithms that require no custom modelling. This is not the case for our deep neural network. Our proposed DNN contains 7 hidden layers and has a dynamic input size based on the feature set. The model will start with all features and use 3 dense layers, halving the number of neurons for each. The structure of the network using feature set 1 can be seen in Figure 9.

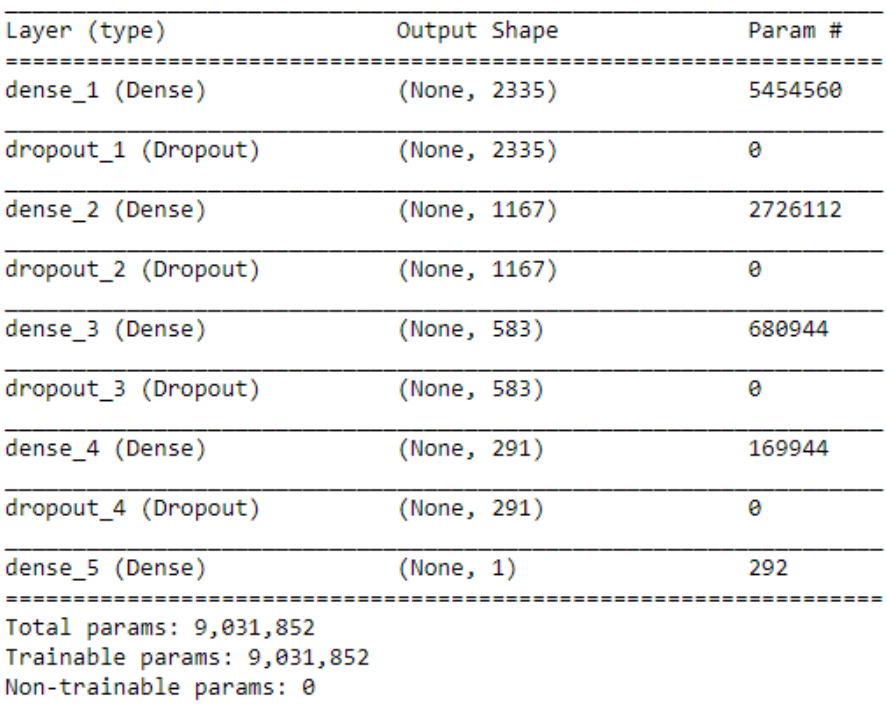
Neural Networks are not categorical in nature, and our model outputs a decimal number between zero and one. When used for classifying during testing, the number is rounded to the closest integer, but does originally return a confidence value.

4.5 Model testing

To evaluate the accuracy of our models, we will be using two test sets. The first one is the included one from EMBER containing 200k test samples. The second dataset includes the 100 most recent PE malware files collected from VirusShare on May 6th 2020 [41]. This second dataset is rather small, but it can still give a basic idea of how effective our models are at detecting modern malware.

4.6 Work and roles

When starting the project, two main roles were assigned, research and programming. Mathias Knutsen had programming as his main responsibility, and Eivind Hestnes Lervik had research. This distribution was only to assign main responsibilities, and to get started on the loose ends. Both have been working closely together combining their knowledge to best produce good results. This



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2335)	5454560
dropout_1 (Dropout)	(None, 2335)	0
dense_2 (Dense)	(None, 1167)	2726112
dropout_2 (Dropout)	(None, 1167)	0
dense_3 (Dense)	(None, 583)	680944
dropout_3 (Dropout)	(None, 583)	0
dense_4 (Dense)	(None, 291)	169944
dropout_4 (Dropout)	(None, 291)	0
dense_5 (Dense)	(None, 1)	292

Total params: 9,031,852
Trainable params: 9,031,852
Non-trainable params: 0

Figure 9: DNN model summary for feature set 1

has been done by using a simplified SCRUM-like development process, where we have scheduled meetings every day, and keep each other posted on own progress. In the end, both have been doing next to equal amounts of both research and programming.

5 Results

5.1 Test results

5.1.1 Deep neural network

Our best performing model overall is our deep neural network, reaching an accuracy of about 95.5%. This accuracy was accomplished using feature set 1. Feature set 4 reached almost 95.4%, a mere 0.1% lower than feature set 1. In terms of recall (TPR), both feature set 1 and 4 are performing close to identical. Metrics and confusion for the model can be seen in Table 2 and Figure 10.

DNN	Accuracy	Precision	Recall	F1 score
Feature set 1	0.955125	0.955412	0.954810	0.955111
Feature set 2	0.948120	0.952044	0.943780	0.947894
Feature set 3	0.947490	0.947696	0.947260	0.947478
Feature set 4	0.953915	0.953140	0.954770	0.953954

Table 2: DNN metrics for all feature sets

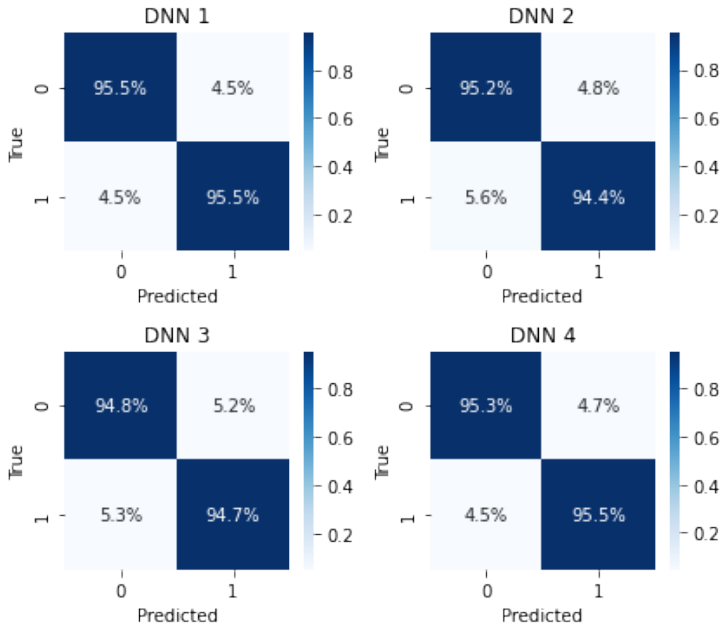


Figure 10: DNN confusion matrix for all feature sets

As seen in Figure 11, feature set 1 and 4 performed the best, with feature set 1 having the advantage on low false positive thresholds, whereas feature set

4 is marginally better, reaching up to 98% at higher thresholds. Feature set 2 and 3 perform similar, and significantly lower than the other two sets.

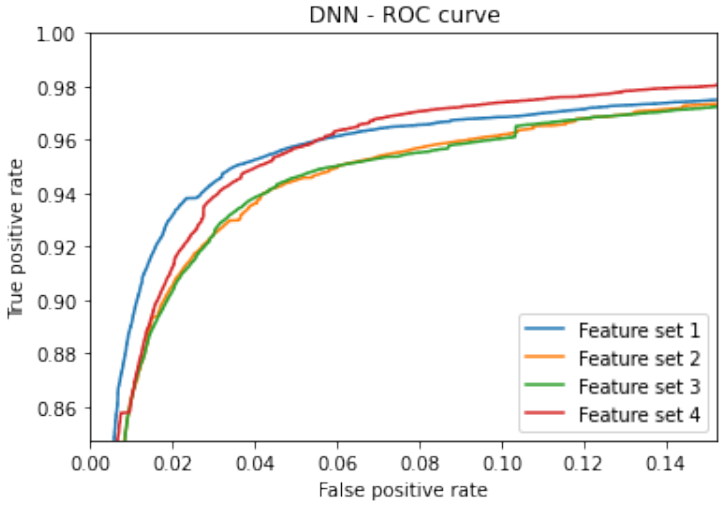


Figure 11: DNN ROC curve for all feature sets

5.1.2 Random forest

Coming in with second highest accuracy at 95.4% is the Random Forest model with feature set 4. Feature set 1 and 3 got an accuracy of just above 95.2%, 0.2% lower than feature set 4. The recall values have similar distributions with almost 94.8% on set 4. Metrics and confusion for the model can be seen in Table 3 and Figure 12.

RF	Accuracy	Precision	Recall	F1 score
Feature set 1	0.952660	0.958678	0.946100	0.952347
Feature set 2	0.939025	0.942009	0.935650	0.938819
Feature set 3	0.952210	0.956473	0.947540	0.951986
Feature set 4	0.954095	0.958946	0.948810	0.953851

Table 3: RF metrics for all feature sets

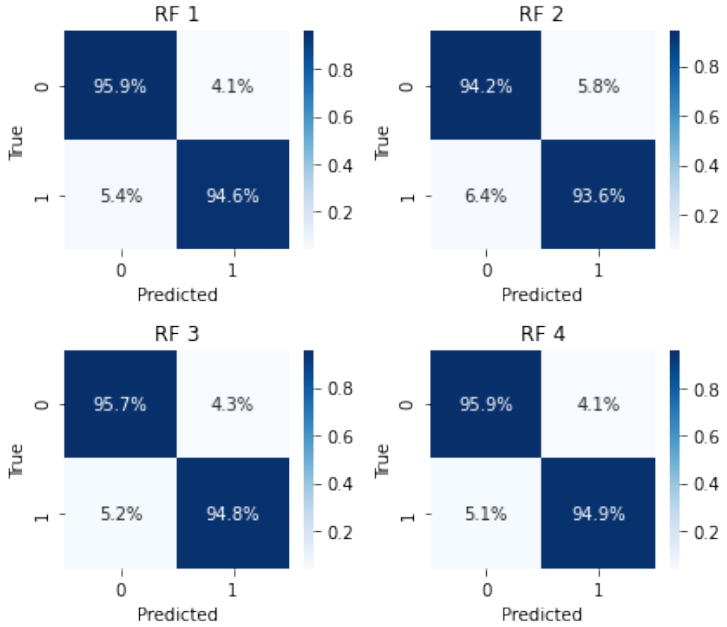


Figure 12: RF confusion matrix for all feature sets

Figure 13 Shows that feature set 1, 3 and 4 are clearly dominating. Set 2 is significantly worse at all FPR thresholds. The best set is arguably feature set 4, with set 1 coming in at a close second place.

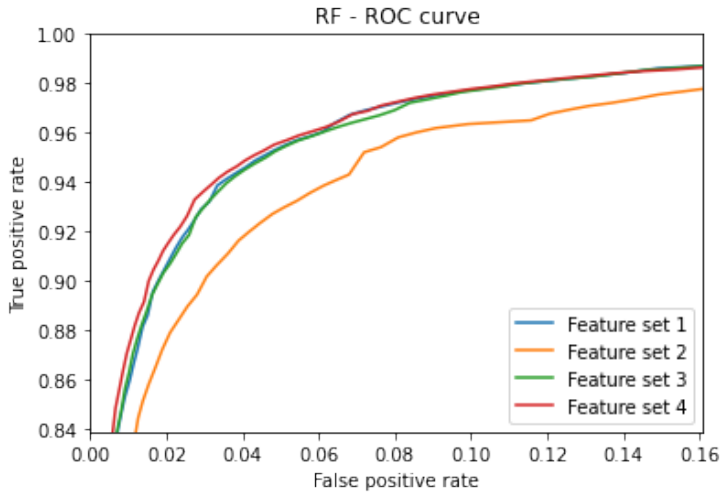


Figure 13: RF ROC curve for all feature sets

5.1.3 Histogram based gradient boosting

Using feature set 1, our HGB model reaches an accuracy of about 93.9%. Once again feature set 4 follows closely with an accuracy of about 93.7%. The recall scores for the two sets are close to identical and outperforms set 2 and 3 by up to 1.5%. It is worth mentioning that the recall scores are significantly higher than the precision scores, meaning that the model correctly classifies more malware at the cost of a higher FPR. Metrics and confusion for the model can be seen in Table 4 and Figure 14.

HGB	Accuracy	Precision	Recall	F1 score
Feature set 1	0.938980	0.925756	0.954510	0.939913
Feature set 2	0.922960	0.902834	0.947940	0.924838
Feature set 3	0.932330	0.921703	0.944930	0.933172
Feature set 4	0.936830	0.922720	0.953520	0.937867

Table 4: HGB metrics for all feature sets

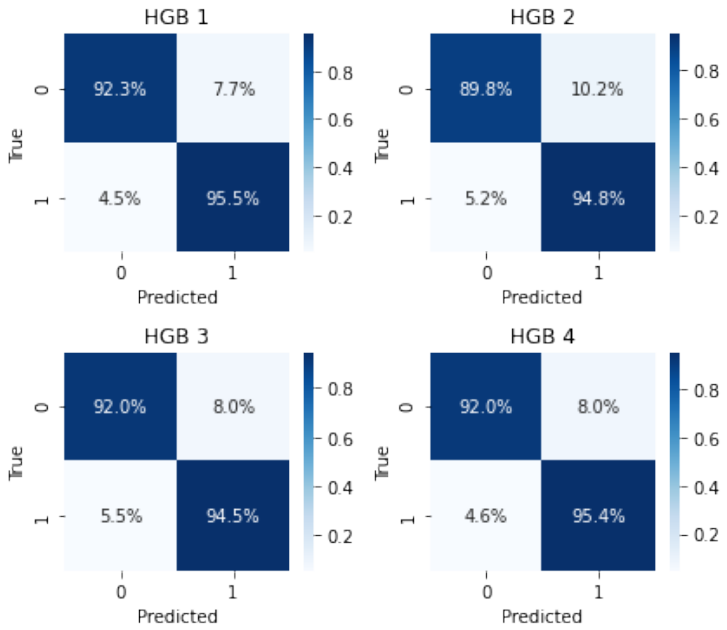


Figure 14: HGB confusion matrix for all feature sets

The ROC curves in Figure 15 shows how feature set 1 and 4 outperforms set 3, and especially set 4 at lower FPR thresholds.

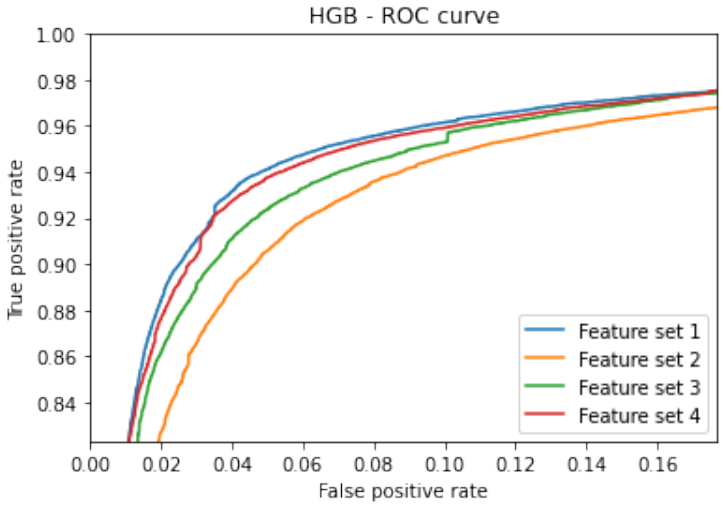


Figure 15: HGB ROC curve for all feature sets

5.1.4 Decision tree

The decision tree model reaches its highest accuracy with feature set 4 at about 92.5%. Feature set 1 is close in accuracy and recall with a difference of 0.2% and 0.1% respectively. Set 2 falls behind with an accuracy of about 89.8%. Metrics and confusion for the model can be seen in Table 5 and Figure 16.

DT	Accuracy	Precision	Recall	F1 score
Feature set 1	0.922205	0.910414	0.936570	0.923307
Feature set 2	0.898070	0.887718	0.911420	0.899413
Feature set 3	0.915680	0.909045	0.923790	0.916358
Feature set 4	0.924960	0.916521	0.935090	0.925713

Table 5: DT metrics for all feature sets

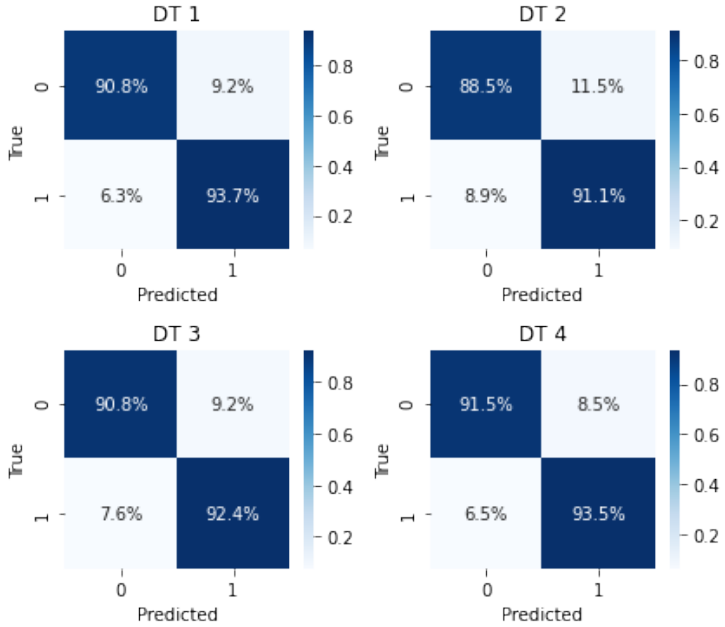


Figure 16: DT confusion matrix for all feature sets

Looking at Figure 17, we see that feature set 4 and 1 are dominating at lower thresholds, while feature set 1 performs better at higher FPR thresholds. Set 2 is performing clearly worse at all thresholds.

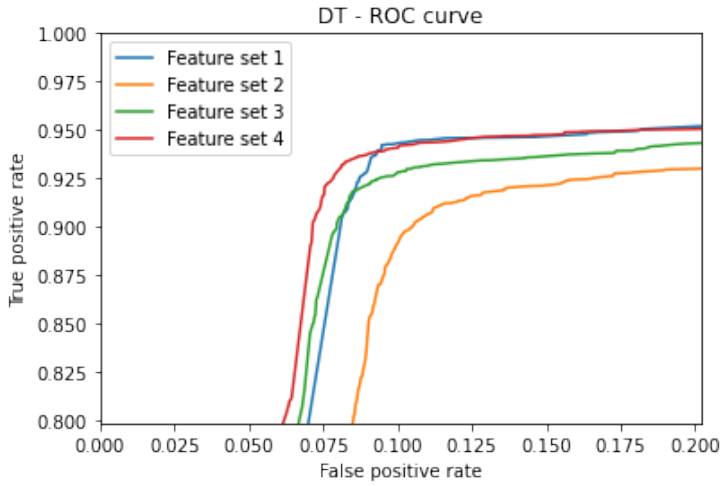


Figure 17: DT ROC curve for all feature sets

5.1.5 Support vector machine

Like the previous models, feature set 1 and 4 are clearly outperforming the two others with accuracies of about 86.3% and 85.1%. The recall score is significantly higher using set 1 at 91.6% versus the 88.7% with set 4. Metrics and confusion for the model can be seen in Table 6 and Figure 18.

SVM	Accuracy	Precision	Recall	F1 score
Feature set 1	0.863165	0.828326	0.916220	0.870059
Feature set 2	0.820395	0.776625	0.899510	0.833563
Feature set 3	0.810210	0.764211	0.897260	0.825408
Feature set 4	0.851020	0.827578	0.886800	0.856166

Table 6: SVM metrics for all feature sets

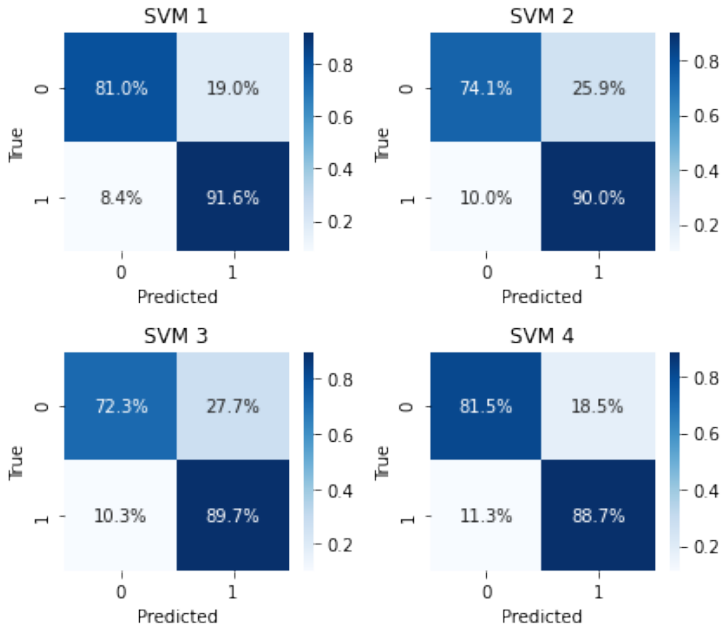


Figure 18: SVM confusion matrix for all feature sets

Figure 19 shows how feature set 1 is dominating with our SVM model. The other sets performs worse with set 2 and 3 as the bottom two.

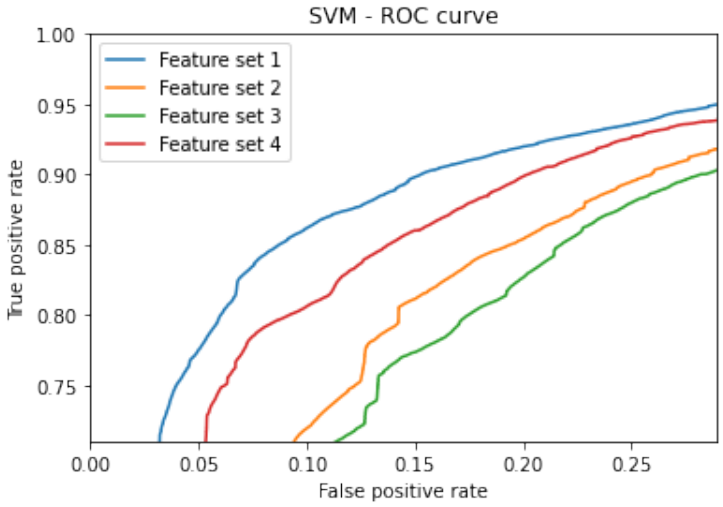


Figure 19: SVM ROC curve for all feature sets

5.1.6 Naive Bayes

The naive Bayes model is a lot worse at predicting than the previous models. The best accuracy obtained was with feature set 2 at 70.8%. Even though the recall score with feature set 1 reached over 90%, the corresponding precision is rather low at 53%, resulting in a low accuracy. Metrics and confusion for the model can be seen in Table 7 and Figure 20.

NB	Accuracy	Precision	Recall	F1 score
Feature set 1	0.550445	0.529648	0.901180	0.667177
Feature set 2	0.708205	0.658699	0.864180	0.747577
Feature set 3	0.621205	0.691172	0.438210	0.536361
Feature set 4	0.685215	0.680372	0.698640	0.689385

Table 7: NB metrics for all feature sets

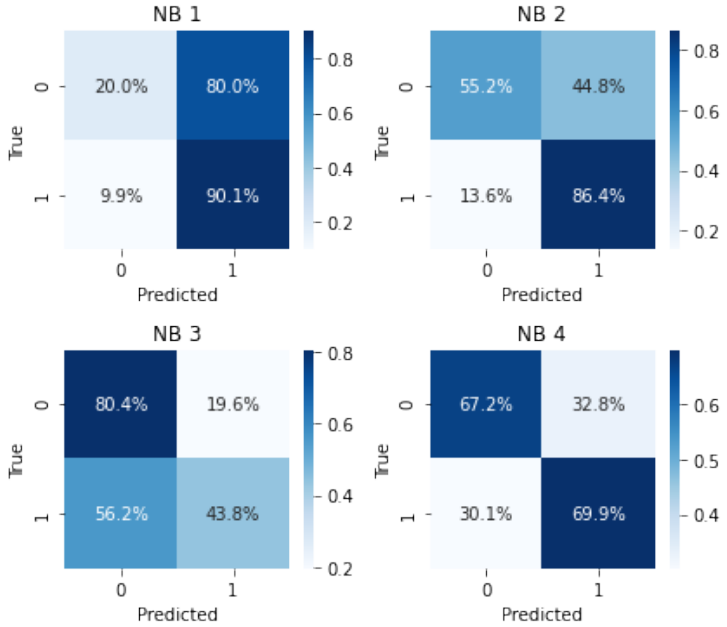


Figure 20: NB confusion matrix for all feature sets

As seen in Figure 21, feature set 1 is performing the worst, while the other three are rather similar in ROC scores.

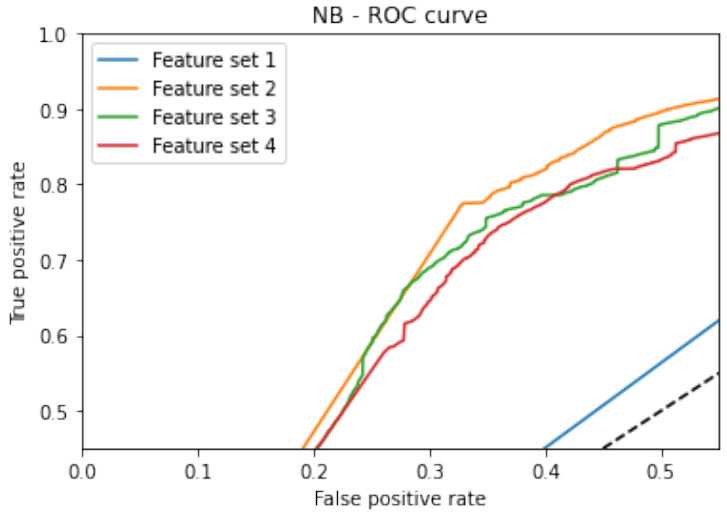


Figure 21: NB ROC curve for all feature sets

5.2 Real world test results

Using our own test set, containing 100 malware samples collected from VirusShare, we achieved the highest detection rate using naive Bayes with feature set 1. The accuracy is in this case same as detection rate (recall), as the test set is unbalanced and contains only malware. The NB model achieved an accuracy of 89%, however, using the average between the different feature sets the model gets 54% accuracy. The second highest accuracy (excluding NB models) at 74% was achieved with our DNN. The average between the DNN feature sets was 71%. RF, HGB, DT and SVM achieved an average accuracy of 61%, 66%, 59% and 50% respectively. All accuracies for the different models and feature sets can be seen in Table 8.

	Feature set 1	Feature set 2	Feature set 3	Feature set 4	Avg.
DNN	0.74	0.70	0.69	0.7	0.71
RF	0.62	0.60	0.61	0.61	0.61
HGB	0.67	0.64	0.67	0.65	0.66
DT	0.41	0.64	0.66	0.66	0.59
SVM	0.48	0.62	0.50	0.41	0.50
NB	0.89	0.75	0.16	0.36	0.54

Table 8: Real world detection rate

5.3 Web solution

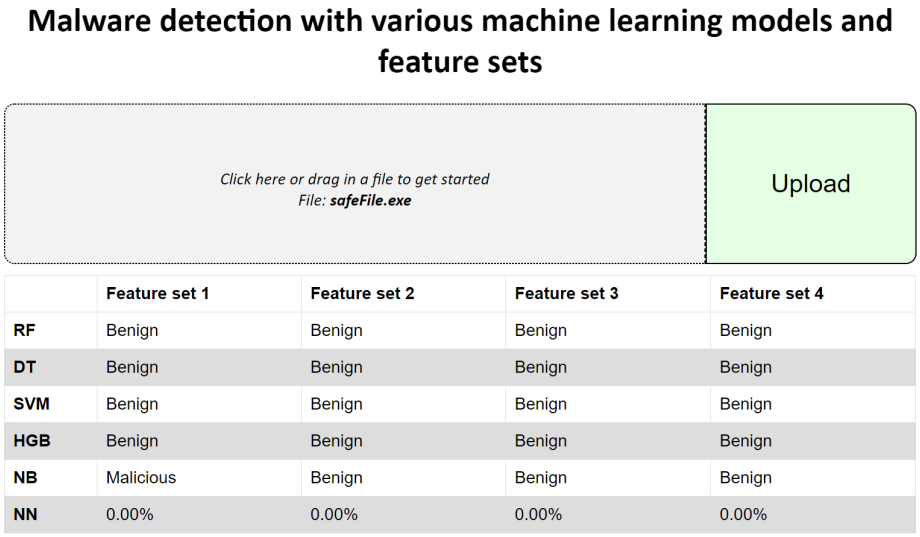
In addition to developing 24 different models for malware detection, we have made a complementary web solution to showcase all the models. The web solution uses the EMBER library to parse a given file, and then makes a prediction using all the 24 models. The interface is a rather simplistic one, that only aims to showcase our models, and not to be a standalone product. A snapshot of the website can be seen in Figure 22. The solution was made with a Python web server, and React as the front-end framework [42].

5.4 Reproducing results

To reproduce our scientific results, please take a look at the system documentation in subsection 9.3.

5.5 Academic engineering and administrative results

Our goal starting this project was to create useful and accurate machine learning models for malware detection that can be used by researchers and developers. Our vision can be found in subsection 9.1. We have achieved a total of 24 different machine learning models, where most perform similar to the reviewed literature. The current state of the system is not considered done, nor is it



	Feature set 1	Feature set 2	Feature set 3	Feature set 4
RF	Benign	Benign	Benign	Benign
DT	Benign	Benign	Benign	Benign
SVM	Benign	Benign	Benign	Benign
HGB	Benign	Benign	Benign	Benign
NB	Malicious	Benign	Benign	Benign
NN	0.00%	0.00%	0.00%	0.00%

Figure 22: Malware detection web solution

incomplete. We have achieved our initial vision, which was to create accurate models ready for further research, optimizations, and development.

When we started this research project as a team of two, we decided in agreement with our supervisor, to not document our progress plan by using a Gannt chart or similar standards. We decided that having frequent meetings would suffice as we were only two people. Our progress is therefore documented in our timetable, meeting notices, meeting minutes and status report. All these documents can be found in the project handbook in subsection 9.4. Because of our approach to planning and guidance from our supervisor, we managed to spread the workload evenly throughout the whole time period.

6 Discussion

6.1 Test results

In this evaluation using the EMBER test set we compared a total of 24 supervised ML approaches to static malware detection. The approaches included 4 different feature sets based on four different methods of feature selection. Looking at the results we see that feature set 1 and 4 are the clear winners, and out-performs set 2 and 3 for every ML model except naive Bayes. As naive Bayes performed significantly worse in the tests than the other models, we choose to not take NB into consideration. Feature set 1 is very similar to the original dataset as it only removes redundant no variance features. The fact that more data gave a better result was expected behavior. As the features from set 4 are extracted from a tree-based model, it is only natural that our tree based models performed good on this feature set. Feature set 1 still yielded slightly better results for most models. Even though feature set 4 was extracted from a tree classifier, it did extraordinary well with the DNN, SVM and NB. Feature set 2 using the chi square test, and feature set 3 using recursive selection did not seem to perform very well for any model except NB. This is possibly because NB was an overall inaccurate model.

The reduction in features from 2351 (set 1) to 675 (set 4) made training the different models exponentially faster. In addition to increased training speeds the feature sets themselves decreased in size. This change in size also decreased the size of some of the trained models. Our DNN ended up at one tenth of its size using feature set 4 compared to set 1. One obvious weakness with reducing the dimensionality of the data is the loss of features with minimal, but some importance. This reduction of features might prevent one from reaching higher accuracies, as the data needed is removed. On the other hand, a good feature set should contain enough features to reach desired accuracies.

For each of the four different feature sets we made six models. Out of these six, the DNN performed the best, reaching over 95.5% accuracy with a detection rate of 95.5%. Based on previous literature we expected our DNN to perform slightly better than the other models, what we did not expect was RF to be as close as it was. Previous works and experiences have shown that RF performs good, but usually not as good as a neural network, however RF still reached an accuracy of 95.4% with a detection rate of 94.9%. We did expect RF to perform better than the DT model, as it is in fact multiple decision trees. The HGB model reached an accuracy of 93.9%, more than one percent lower than RF, but with a recall of 95.5%. This detection rate is as high as our DNN. As HGB is based on LightGBM, we expected these results to be close to the top. The fact that SVM and naive Bayes performed worse was as expected. SVM was running with a linear kernel, and naive Bayes performed poorly in the previously reviewed literature. Looking at our requirements documentation in subsection 9.2, we set out to meet two main requirements. The first was for any future developer to be able to utilize our models to successfully classify malware, which we indeed have achieved. The second requirement which we

also achieved was to present future researchers with a variety of models, where they can compare performance and continue improving the models.

6.2 Real world test results

Looking at the averages from Table 8, we see that the average accuracy for our NB model is a lot lower than the highest accuracy. The reason why NB got such a high detection rate can be seen in Table 7. We see that the recall score for feature set 1 is over 90% and would expect this detection rate on the real-world tests. This is indeed the case; however its precision is only at about 52%, meaning that the amount of false positives is very high. The NB method is probably not suitable for malware detection. The DNN is achieving over 70% accuracy on average and is performing the best both in the EMBER and real world tests. It is not surprising to see that HGB beat RF on average, as HGB had a higher recall score with the EMBER test set.

These results are not clear indications for what type of model works best, but they give a good idea of how our models would perform on unseen and modern PE files. This test set only contains 100 entries, and the malware type is random. A never-before-seen malware, a zero-day, will never be detected by standard signature-based methods. Having a 70% chance of detection is much better than guessing or just letting it execute.

6.3 Overall perspective and professional ethics

With the rise of technology usage in the information age, more people than ever before now has access to the internet. Many important services like banking, shopping and entertainment are now happening online. An ever-increasing amount of information is being shared and stored online. All this has made the internet a lucrative hunting ground for people with malicious intent. As a result, vast amounts of malware are being created and distributed. A malware attack can cause damage to individual computers, as well as large networks and servers, causing both economic and social impact. Preventing such attacks from happening is the right thing to do and can therefore have a positive effect on society as a whole, and as such we hope that our results can help push the envelope in the everlasting battle against malware.

When publishing this type of research, one of the possible ethical problems to consider is that the published work can be utilized by malware creators, to create malware that is harder to detect. However, we feel that publishing the work will benefit the greater good, even though a few people might have bad intentions. Another thing to consider when creating any kind of software, is the environmental impact. Heavy processing requires a lot of energy in the form of electricity. Much of the world's electricity is generated through the burning of fossil fuels. As a result, one should always strive to make code as efficient as possible. This is part of the reason we chose static as opposed to dynamic analysis for our project.

When referring to professional ethics for software engineering, one often thinks about privacy and the processing of personal data. Rather than providing a traditional product for the everyday user, we provide research and a variety of machine learning models to detect malware. Our research is based on harmful and malicious software samples. When working with such content, it is important to always take the necessary precautions. When hosting or distributing these types of malware, one does not consider the public good. To counter this problem, only the parsed features from the malware samples are distributed. This way the public can only benefit from the data.

6.4 Academic engineering and administrative results

As described in the results chapter we managed to achieve our initial goal. We were initially slightly concerned about only using frequent meetings for planning and distributing workloads over time. This ended up working to our advantage and made a much more flexible plan. Instead of sticking to a fixed plan, we got regular input from our supervisor and managed to distribute the workload as well as tweaking our future plans as we went. Our SCRUM-like approach also optimized the workflow between both of us.

6.5 Working as a group

Working together as a small team of two has been a positive experience. We have both previously worked in teams of four or more people. This experience has been completely different. The communication has been a lot more fluent, and the workflow has been well above average. One disadvantage with being two people are conflicting opinions. Any conflicts than occurred had no immediate input from a third party. This was simply solved by asking our supervisor.

7 Conclusion and further work

7.1 Conclusion

As outlined in the introduction of our thesis (subsection 1.4), we set out to compare different supervised machine learning approaches to static malware detection. We compared six different ML models and four different sets of features. Out of the four sets of features, the variance-based and tree-based sets performed the best. Using a smaller part of the original data, like the tree-based feature set, yield very similar results with the advantage of faster training speeds and lower space usage. In the end what works best is dependent on what one aims to achieve, higher accuracies or faster model training.

Out of the six models, naive Bayes, support vector machines, and decision trees performed at a lower level than the rest. The algorithms behind these three are rather simple and does not perform as good as the other three models. The deep neural network comes in at first place counting both accuracy and detection rate. Random forest is close behind with its high overall accuracy score, but so is the histogram based gradient boosting model with its high detection rate. All three models performed good for malware detection and could possibly be improved by further work and research.

7.2 Further work

Although we achieved good results with our proposed models, our priority was to implement different machine learning models to compare, rather than spending too much time tweaking to achieve marginal gains. Machine learning has a lot of room for experimenting and tuning. It is very possible that even better results could be achieved by tweaking the proposed solutions. Windows is not the only operating system that is targeted by malware. A possible future improvement could be creating a solution that works with multiple executable formats. Microsoft's PE format is relatively similar to the ELF format used by Linux executables, and a uniform machine learning model should therefore not be impossible to achieve. Another thing to keep in mind is that as malware detection get cleverer, so does the malware. It is a perpetual battle where the malware keeps changing. As such, a solution that works good today, might not work good in the future. This is highlighted by the literature as well as our real-world tests.

8 References

[1] Robert Moir. *Defining Malware: FAQ*. https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10), Accessed May 9, 2020. 2009.

[2] William Stallings et al. *Computer security: principles and practice*. Pearson Education Upper Saddle River, NJ, USA, 2012.

[3] Mike Barwise. *What is an internet worm?* http://www.bbc.co.uk/webwise/guides/internet-worms, Accessed May 9, 2020. 2010.

[4] Per Christensson. *Trojan Horse Definition*. https://techterms.com/definition/trojanhorse, Accessed May 9, 2020. 2006.

[5] Per Christensson. *Ransomware Definition*. https://techterms.com/definition/ransomware, Accessed May 9, 2020. 2019.

[6] John Von Neumann, Arthur W Burks, et al. "Theory of self-reproducing automata". In: *IEEE Transactions on Neural Networks* 5.1 (1966), pp. 3–14.

[7] William WS Chen. *Statistical methods in computer security*. CRC Press, 2004.

[8] Deborah Russell et al. *Computer security basics*. " O'Reilly Media, Inc.", 1991.

[9] FBI Gov. *The Morris Worm*. https://www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218, Accessed May 9, 2020. 2018.

[10] James Griffiths. *'I love you': How a badly-coded computer virus caused billions in damage and exposed vulnerabilities which remain 20 years on*. https://edition.cnn.com/2020/05/01/tech/iloveyou-virus-computer-security-intl-hnk/index.html, Accessed May 9, 2020. 2020.

[11] BBC News. *More than 100 arrests, as FBI uncovers cyber crime ring*. https://www.bbc.com/news/world-us-canada-11457611, Accessed May 9, 2020. 2010.

[12] Ellen Nakashima and Joby Warrick. *Stuxnet was work of U.S. and Israeli experts, officials say*. https://www.washingtonpost.com/world/national-security/stuxnet-was-work-of-us-and-israeli-experts-officials-say/2012/06/01/gJQAlnEy6U_story.html, Accessed May 9, 2020. 2012.

[13] Gregg Keizer. *Is Stuxnet the 'best' malware ever?* https://www.infoworld.com/article/2626009/is-stuxnet-the--best--malware-ever-.html, Accessed May 9, 2020. 2010.

[14] Leo Kelion. *Cryptolocker ransomware has 'infected about 250,000 PCs'*. https://www.bbc.com/news/technology-25506020, Accessed May 9, 2020. 2013.

[15] Jonathan Berr. *"WannaCry" ransomware attack losses could reach $4 billion*. https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/, Accessed May 9, 2020. 2017.

[16] The Cylance Team. *How Traditional Antivirus Works*. https://threatvector.cylance.com/en_us/home/how-traditional-antivirus-works.html, Accessed May 9, 2020. 2017.

[17] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[19] Michael A Nielsen. *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA, USA: 2015.

[20] strikingloo. *Convolutional Neural Networks: Training an Image Classifier with Keras*. https://www.datastuff.tech/machine-learning/convolutional-neural-networks-an-introduction-tensorflow-eager/, Accessed May 12, 2020.

[21] Christopher Olah. *Conv Nets: A Modular Perspective*. http://colah.github.io/posts/2014-07-Conv-Nets-Modular/, Accessed May 14, 2020. 2014.

[22] Kim Jinsol. *What is Convolution Neural Network?* https://gaussian37.github.io/dl-concept-cnn/, Accessed May 16, 2020. 2018.

[23] Denny Britz. *Introduction to RNNs*. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/, Accessed May 12, 2020. 2015.

[24] R Vinayakumar et al. "Robust intelligent malware detection using deep learning". In: *IEEE Access* 7 (2019), pp. 46717–46738.

[25] H. S. Anderson and P. Roth. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models". In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.04637 [cs.CR].

[26] Mohamed Belaoued and Smaine Mazouzi. "A real-time pe-malware detection system based on chi-square test and pe-file features". In: *IFIP International Conference on Computer Science and its Applications*. Springer. 2015, pp. 416–425.

[27] Eric W. Weisstein. *"Chi-Squared Test." From MathWorld–A Wolfram Web Resource*. https://mathworld.wolfram.com/Chi-SquaredTest.html, Accessed May 9, 2020. 2017.

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

[29] Lakshmanan Nataraj et al. “Malware images: visualization and automatic classification”. In: *Proceedings of the 8th international symposium on visualization for cyber security*. 2011, pp. 1–7.

[30] Sugandha Lahoti. *Malware Detection With Convolutional Neural Networks in Python*. `https://dzone.com/articles/malware-detection-with-convolutional-neural-networ`, Accessed May 9, 2020. 2018.

[31] Jhu-Sin Luo and Dan Chia-Tien Lo. “Binary malware image classification using machine learning with local binary pattern”. In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 4664–4667.

[32] Timo Ojala, Matti Pietikäinen, and David Harwood. “A comparative study of texture measures with classification based on featured distributions”. In: *Pattern recognition* 29.1 (1996), pp. 51–59.

[33] How Malware Evades Detection by Sandboxes. “Evasive Malware Tricks”. In: ().

[34] Ivan Firdausi, Alva Erwin, Anto Satriyo Nugroho, et al. “Analysis of machine learning techniques used in behavior-based malware detection”. In: *2010 Second international conference on advances in computing, control, and telecommunication technologies*. IEEE. 2010, pp. 201–203.

[35] Ishai Rosenberg and Ehud Gudes. “Bypassing system calls–based intrusion detection systems”. In: *Concurrency and Computation: Practice and Experience* 29.16 (2017), e4023.

[36] Matilda Rhode, Pete Burnap, and Kevin Jones. “Early-stage malware prediction using recurrent neural networks”. In: *computers & security* 77 (2018), pp. 578–594.

[37] Microsoft. *PE Format*. `https://docs.microsoft.com/en-us/windows/win32/debug/pe-format`, Accessed May 9, 2020. 2019.

[38] Wikipedia user Finnusertop. *PE EXE file format's header*. `https://upload.wikimedia.org/wikipedia/commons/1/1b/Portable_Executable_32_bit_Structure_in_SVG_fixed.svg`, Accessed May 9, 2020. 2019.

[39] Romain Thomas. *LIEF - Library to Instrument Executable Formats*. https://lief.quarkslab.com/. 2017.

[40] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[41] *VirusShare*. `https://virusshare.com/`, Accessed May 9, 2020.

[42] ReactJS. *A JavaScript library for building user interfaces*. `https://reactjs.org/`, Accessed May 15, 2020. 2020.

9 Attachments

73

Malware detection with machine learning Vision document

Version 1.2

Revision history

Date	Version	Description	Author
20/01/2020	1.0	Initial vision	Mathias Knutsen and Eivind Hestnes Lervik
11/05/2020	1.1	Audits changing our vision to better fit our task.	Mathias Knutsen and Eivind Hestnes Lervik
12/05/2020	1.2	Widening the scope of the product	Mathias Knutsen and Eivind Hestnes Lervik

Table of contents

1. Introduction

This project is mainly research based and therefore doesn't provide a proper system as a product, but rather various machine learning models based on results from our research. In addition, we provide a supplementary web-based prototype system showcasing the models. These two components make up our product as a whole.

2. Summary - problem and product

2.1 Problem summary

Problem with	detecting unseen malware
affects	everybody who uses a (Windows) computer
as a result	computers are unnecessarily infected by malware
a successful solution will	detect malware before it infects a computer, and decrease overall infections

2.2 Product summary

For	Windows users, researchers, developers
that	encounter malware
produktet navngitt	malware detection with machine learning
that	detects malware
unlike	traditional signature-based malware detection
our product have	machine learning models that can detect unseen malware

3. Overall description of stakeholders and users

3.1 Stakeholder summary

Name	Description	Role
Researchers	Other machine learning researchers can benefit from reading our results.	No particular role.
Developers	Developers can use the models and results to create their own antimalware software	No particular role.

3.2 User summary

Name	Description	Role	Represented by
Researchers	Other machine learning researchers can benefit from reading our results.		Themselves

Developers	Developers can use the models and results to create their own antimalware software		Themselves

3.3 User environment

To understand or be able to use the models and prototype, a research or developer environment is required.

3.4 Summary of user needs

Needs	Priority	Affects	Today's solution	Suggested solution
Useful and accurate machine learning models that can be used for research or in a system.	1	The ability to accurately detect new unseen malware.	Signature based anti-malware.	Malware detection using machine learning.

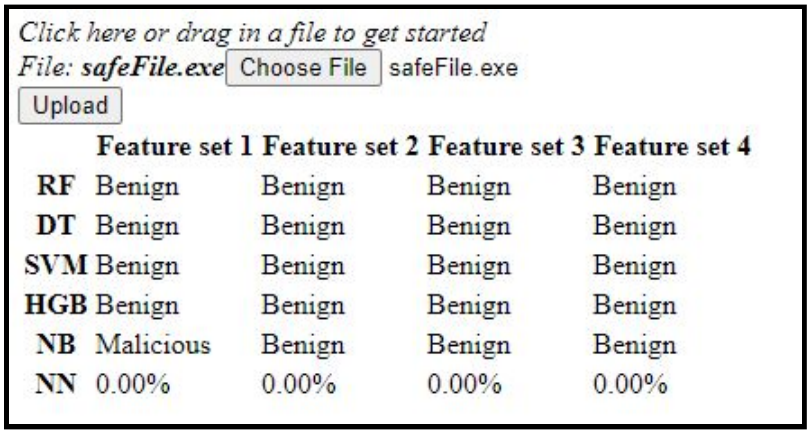
3.5 Alternatives to our product

Some antivirus software today does use a combination of machine learning and signature based approaches, however most are signature based. The ones that do use machine learning are usually behaviour analysis, which can be very resource heavy.

4. Product overview

4.1 The product role in the user environment

We cannot sketch machine learning models, but the below sketch of the planned web prototype shows the various models that we would like to create.



4.2 Prerequisites and dependencies

While the planned web solution can be utilized by anyone, however it is not intended as a fully fledged product. Our target is individuals with a computer research or developer background. The only

prerequisite is therefore the background knowledge to utilize the research and proposed models.

5. Functional properties of the product

- The web solution must have the ability to upload and classify any executable windows file.
- The proposed models must be able to accurately decide if a file is malicious or benign.

73

Malware detection with machine learning
Requirements document

Version 1.1

Revision history

Date	Version	Description	Author
11/05/2020	1.0	Initial	Mathias Knutsen and Eivind Hestnes Lervik
12/05/2020	1.1	Widening the scope of the product	Mathias Knutsen and Eivind Hestnes Lervik

Table of contents

1. Introduction

The purpose of this document is to describe the functional requirements for our product. As mentioned in our vision document, our product is merely a variety of machine learning models, and a supplementary web solution to showcase the different models.

2. User stories

As a developer
I wish to classify a given executable, and find out if it is a malware or not
so that I can develop an anti-malware solution.

Acceptance criteria:
If I am a developer I have the option to use different machine learning models to classify malware.

As a researcher
I wish to compare different machine learning models classifying a given executable
so that I can satisfy my curiosity and potentially continue researching/developing the models.

Acceptance criteria:
If I am a user or researcher, I can compare multiple machine learning models, satisfy my neverending curiosity and be able to tweak the models.

3. Prototype

We will not be prototyping the machine learning models, however, the illustration below is a dummy prototype of the supplementary web-solution.

Malware detection with various machine learning models and feature sets

Click here or drag in a file to get started
*File: **safeFile.exe***

Upload

	Feature set 1	Feature set 2	Feature set 3	Feature set 4
RF	Benign	Benign	Benign	Benign
DT	Benign	Benign	Benign	Benign
SVM	Benign	Benign	Benign	Benign
HGB	Benign	Benign	Benign	Benign
NB	Malicious	Benign	Benign	Benign
NN	0.00%	0.00%	0.00%	0.00%

73

Malware detection with machine learning
System document

Version 1.1

Revision history

Date	Version	Description	Author
11/05/2020	1.0	Initial	Mathias Knutsen and Eivind Hestnes Lervik
12/05/2020	1.1	Widening the scope of the product	Mathias Knutsen and Eivind Hestnes Lervik

Table of contents

1. Introduction

In this document we will describe the development part of our project, including architecture, file structure, installation and more. Our project is mainly based on research, which includes both literature and code that tests different machine learning method's ability to detect malware. The purpose of this document is to guide anyone who wants to reproduce our research results. Any details about the different machine learning models can be found in the main report.

2. Architecture

Our code consists of two main components:

- Machine learning models doing malware detection in Python
- Web solution
    - Python webserver predicting malware using the created models.
    - Client website communicating with the Python webserver

3. Project structure

Machine learning models

./ember2018/	% Includes the ember dataset **(Not included in attachments)**
./models/*.bin	% Includes trained models
./preprocessed/*.bin	% Includes modified training data
./CustomDNN.ipynb	% Proposed Deep Neural Network
./CustomDT.ipynb	% Proposed Decision Tree
./CustomHGB.ipynb	% Proposed Gradient Tree Booster
./CustomNB.ipynb	% Proposed Naive Bayes
./CustomRF.ipynb	% Proposed Random Forest
./CustomSVM.ipynb	% Proposed Support Vector Machine
./Evaluation.ipynb	% Calculates metrics for all ML methods above
./Preprocessing.ipynb	% Uses the ember dataset to create four smaller datasets
./RealWorldTesting.ipynb	% Calculates metrics for all ML methods above using real world malware
./RealWorldDataset.bin	% Our own dataset

The above list shows the general project structure of our machine learning models. The dependencies are listed under "Installation and running".

Python web-server and client

./app.py	% The web server using models to predict any Windows executable
/www/*.*	% The client build files
/react/*.*	% The client source files - React project[1]

The above list shows the structure of our web-server and client. The client has it's raw source code in /react, and the built code inside /www. As the website is only supposed to be a demo in addition to our main report, we will not be providing any documentation about building the react app.

4. Server-client

Our very simple web-server have the following endpoints:

- /*.* % Returns static files
- /upload % Returns machine learning predictions based on file

5. Security

As our web server is once again only a small addition to show the potential of our research, we have not made any specific security measures. We intend the webserver to be run locally, and therefore do not require any certificates or anti-hacking measures.

6. Installation and running

Note: *Before proceeding please note that this was run on a virtual linux machine with 32 cores and 32 gigabytes of memory*. *The website has been tested and works on Windows.*

Prerequisites:

- Python (Version 3.6.8 or 3.6.9)
- pip
- git

List of python libraries used:

- numpy % Math library
- flask % Web server
- flask_cors % Web server CORS
- tensorflow % A machine learning library, specializing in neural networks
- joblib % Save and load Python objects
- matplotlib % Draw figures
- scikit-learn % A machine learning library, specializing in classification
- seaborn % Draw figures
- jupyter % Notebook tool for writing and running python code
- keras % High level tensorflow
- EMBER[2] % This is not the dataset, this is a library containing tools for it's usage

[1] https://reactjs.org/, A JavaScript library for building user interfaces, last accessed May 11, 2020.
[2] https://github.com/endgameinc/ember, EMBER, Anderson, H.S. and Roth, P., last accessed May 11, 2020.

Notes

To minimize the attachment zip size, we have omitted the training and test data. This means that to reproduce our results, one needs to download the EMBER dataset, and preprocess the data. Note that the compiled models are in fact included so that the website works out of the box.

Installation

1. Make sure you have installed the prerequisites
2. Download attachments zip folder including all source code (About 0.5GB).
3. Run “pip install -r requirements.txt”
4. ***The following steps are only for those who wish to reproduce our results and spend time training and evaluating the models. Those who wish to run the website can skip to the “Run Website” chapter.***
5. Download the EMBER dataset from their github at https://github.com/endgameinc/ember.
6. Navigate to *./notebooks/*
7. Run “jupyter notebook”, and open up preprocessing.ipynb, and change “data_dir” to wherever you downloaded the ember dataset
8. Run preprocessing.ipynb *

Run models *

1. All models can be run in jupyter notebook. See files named “Custom*.ipynb” *
2. Evaluation.ipynb can be run after preprocessing and model training is done.

Run Website

1. Navigate to *./website/*
2. Execute “python app.py” (Might need sudo/admin permissions, see point 4)
3. Wait for the application to start
4. Visit localhost:port to see the website (The port might need to be changed depending on what the system allows. Currently set to 80)

* *Might take hours or even days depending on your system*

7. Source code documentation

All our code is documented with inline comments. This is the usual go-to when making machine learning models with python.

Project handbook

Table of contents

Meeting notices with minutes

All meeting notices with its corresponding minutes are listed below.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 1

Date: 14/01/2020
Time: 15:30
Place: Berg: F-S040B (MAZEMAP)

Topics

- Meeting schedule
- Partitioning project tasks
  - Dataset
  - Literature review
  - Amount
  - Feature-sets
  - Clustering
  - Classification
  - Approaches
- Progress plan – GANNT?
- Choice of development process
- Documentation requirements
- Timetables and weekly reports
- Choice of language
- Minutes from this meeting will be sent via email.

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 1

January 14, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the first meeting between the team and the supervisor about this project. The team and the supervisor agreed that meeting once a week, at least in the beginning would be a good idea and the next meeting was scheduled for next Tuesday 15:00 at the same location. Changes can occur if necessary.

As for the exact details of what the project will include, the team will look at earlier works and available data and find out what can be done. Specific details of the projects will be discussed later. Malware datasets are somewhat hard to find and are often unbalanced, but the team will try to find suitable dataset(s) before the next meeting.

The project will consist of several different parts including a literature review where the team will look at articles and work done by others and see what results they have achieved. Articles and datasets with code results can be found at websites like Kaggle or Google Scholar. When the team have some idea of what work they will do, we can decide if we need to utilize specialized hardware. Depending on what the team find and choose to do, feature sets etc. is maybe not such a big deal, this can be discussed later. The team can look at the literature and possibly test themselves to find out which approaches works best. Maybe the team can create a website.

How the team organizes is not such a big deal, as long as they manage to get the work done. Since the team is so small the type of development process is not were relevant. What kind of documentation that is needed will be looked at later. Everyone agreed that English would be the language of choice for this project.

Final words from the supervisor was that the team should read some articles and find useful datasets. Look at how malware is typically detected, particularly signature-based detection.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 2

Date: 28/01/2020
Time: 15:00
Place: 242 Sydfløy

Topics

- Look at minutes from last meeting
- Literature Review of recent works
  - Platform: Windows, as there already is so much research for android.
  - How many papers?
  - How much?
- Dataset: Ember
  - Static
- Vision Document
  - Angle it towards making a “Malware Detection Product”?
- VT1 TDAT3001 – 07. February – Workshop presentation

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 2

January 28, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the second meeting between the team and the supervisor about this project. The next meeting will happen the 18th of February at 15:00.

We started of discussing the dataset and agreed to further research this in the coming weeks. We will look at encrypted payloads and find sources that cite the Ember dataset.

Further research that can be done includes how anti-virus software work and how they can improve.

Up to 50% of the report can be literature review and malware history.

We agreed to only document more complex machine learning methods as some knowledge about ML is expected upon reading the final report.

Morrison also added that we can look at Cylance.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 3

Date: 18/02/2020
Time: 15:00
Place: 242 Sydfløy

Topics

- Look at minutes from last meeting
- Status Report
  - VM
  - Main Report Setup
- Supervised vs unsupervised
- Others

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 3

February 18, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the third meeting between the team and the supervisor about this project. The next meeting will happen the 3rd of March at 15:00.
The minutes from the previous meeting is approved.
The team gives a short status report of the work that has been done since the previous meeting.
The team suggests that part of the task can be to create some kind of web solution, for example where you can upload an executable file and the website will tell you whether it thinks it is malware or not.
The supervisor is positive to the idea.
The team ask whether they should focus on supervised or unsupervised learning. The supervisor says they can look at both and experiment with different things and then decide what they want to focus on.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 4

Date: 03/03/2020
Time: 15:00
Place: 242 Sydfløy

Topics

- Look at minutes from last meeting
- Status report
  - Main report
  - Custom CNN
- Decide next meeting – exam 18.03

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 4

March 3, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the fourth meeting between the team and the supervisor about this project. The next meeting will happen the 24th of March at 15:00.
The minutes from the previous meeting is approved.
Most of the meeting was the team giving a in depth status report of the work that has been done since the previous meeting.
After the status report Donn Morrison pointed out that we should reduce the number of features for classification by using PCA/Greedy first selection. Scikit Learn should have the tools needed to perform this reduction of features.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 5

Date: 28/04/2020
Time: 15:00
Place: Jitsi Meet

Topics

- Status report
  - Feature selection
  - ML models (DNN, RF, SVM, DT), possibly others
  - Web server with demonstration
- Questions
- Decide next meeting

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 5

April 28, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the fifth meeting between the team and the supervisor about this project. The next meeting will happen the 12th of May at 15:00.

The minutes from the previous meeting is approved.

The team gives a status report of the work that has been done since the previous meeting.

The team asked what more could be added to the proposed web solution. The supervisor suggested that if a kNN-model were to be used the team could investigate displaying the files from the dataset that are most similar (nearest neighbor) of the input file. The team said they would investigate it.

The team and supervisor agreed that the team should send a draft of the report within a week so that the supervisor can see the progress and comment.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 6

Date: 12/05/2020
Time: 15:00
Place: Jitsi Meet

Topics

- Status report
    - Report progress
    - Documentation progress
- A few questions
- Decide next meeting

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 6

May 12, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the sixth meeting between the team and the supervisor about this project. The next meeting will happen the 18th of May at 15:00.

The minutes from the previous meeting is approved.

The team gives a status report of the work that has been done since the previous meeting.

The supervisor suggests that the documentation should be answered with the ML-models as the product, rather than just the prototype web solution.

Discussion regarding the reports format has been cleared up.

The team should create a video presentation of the project.

The team asks whether a conclusion that summarizes the literature review should be added. The supervisor says that it should.

The supervisor suggests that the website should work “out of the box” without having to train the models, so that it is easier to test it.

Knutsen, Mathias
Lervik, Eivind Hestnes
Morrison, Donn

Meeting notice 7

Date: 18/05/2020
Time: 15:00
Place: Jitsi Meet

Topics

- Status report
    - Report progress
    - Documentation review
- A few questions

Regards,

Mathias Knutsen and Eivind Hestnes Lervik

Meeting minutes 7

May 18, 2020

Present

Donn Morrison, Supervisor

Mathias Knutsen

Eivind Hestnes Lervik

Agenda

This is the sixth meeting between the team and the supervisor about this project. This was the last meeting before the project is finished.

The minutes from the previous meeting is approved.

The team gives a status report of the work that has been done since the previous meeting.

There is not necessary with an extra summary in addition to the abstract.

It is fine that literature review is its own chapter, so that the final report has 7 chapters instead of 6.

The team should present some title suggestions to the supervisor so that he can help them choose.

Camera video of the team is not necessary for the presentation video.

The abstract should be included in the final report as well as added to the Inspera submission.

Status report

Due to certain circumstances we decided to write a status report instead of having a meeting.

Status report as of 23.03.20

What has been done

We have been working on writing the literature review part of the assignment. Looking at some of the related articles we had found earlier, we wrote about the technique of converting a malware binary to an image and using a Convolutional Neural Network to classify it. Having previously been able to train a neural network on the Ember dataset, we have since worked on using different methods, such as Random Forest (classification). We have also now implemented the ability to classify any PE-file using the trained model. To get access to malware PE-files for testing, we contacted VirusShare (https://virusshare.com/), and received access to their database of malware. In addition to that, we have written more on the other parts of the report, such as the supporting literature. This includes a quick review of the history of malware, information about the Ember dataset, different types of malware and why machine learning could be the future in the battle against malware.

What will be done next

- Look at other available classification methods
- Further work on the literature review
- Further work on feature reduction
- General optimizations
- Website (not prioritized yet)

Extending to UNIX-based systems, PE vs ELF (?)

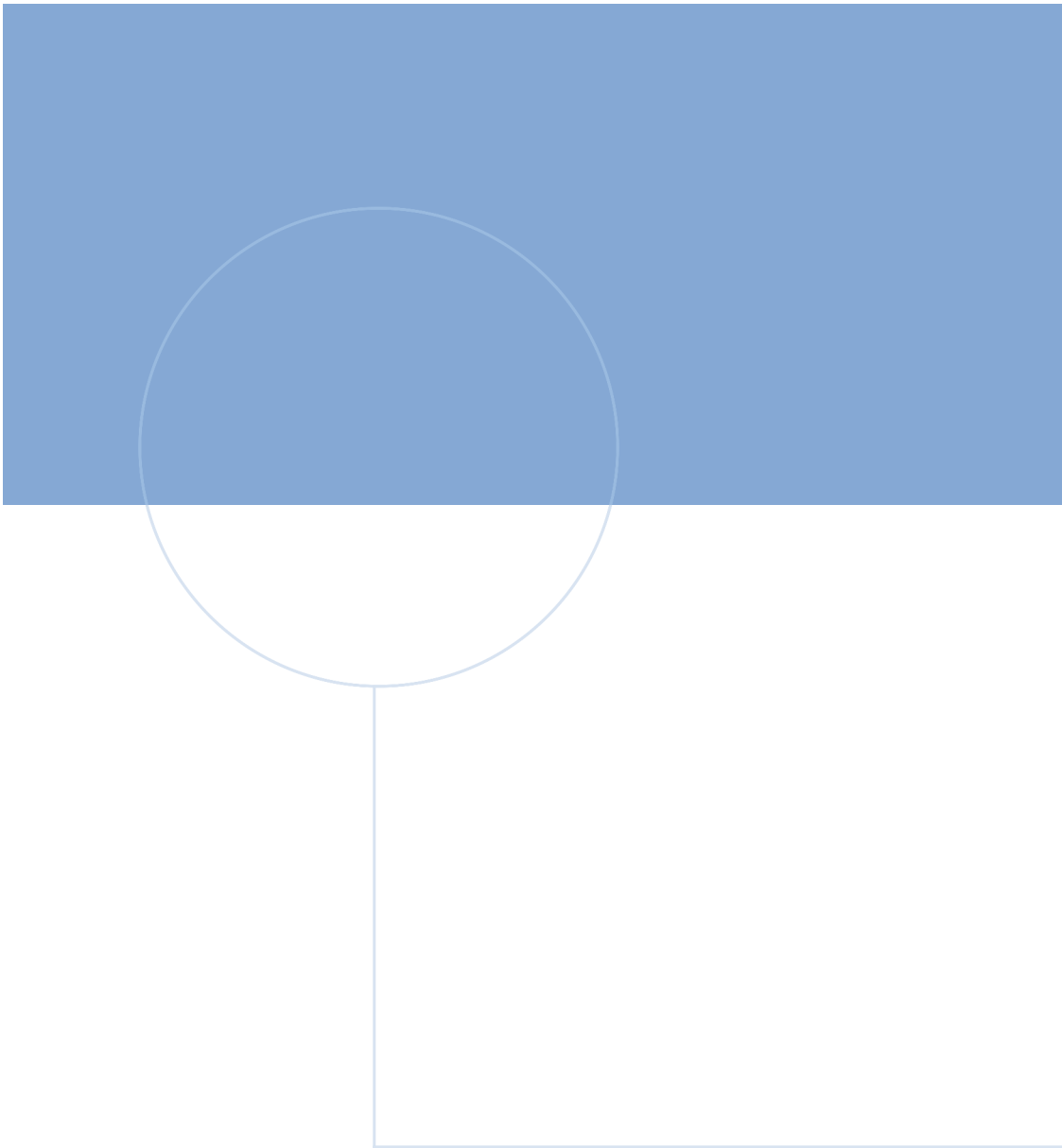
Timesheet

Week	Activity	Mathias	Eivind	Name	Sum	
Week 2	General Planning	16	16	Mathias	507	*hours*
Week 3	**Meeting 1**	1	1	Eivind	506	*hours*
	File structure setup	1	0	Total	1013	*hours*
	Featureset research	8	8	Required	1000	*hours*
	General research	16	16	**Completion**	**101,3**	***%***
Week 4	General research	11	11			
	Look into making website	4	0			
Week 5	**Meeting 2**	1	1			
	General research	10	16			
	Programming	18	6			
Week 6	General research	4	13			
	Programming	20	8			
Week 7	General research	12	13			
	Programming	2	3			
Week 8	**Meeting 3**	1	1			
	General research	9	16			
	Programming	7	3			
Week 9	General research	15	20			
	Programming	15	6			
Week 10	**Meeting 4**	1	1			
	General research	8	24			
	Programming	12	6			
Week 11	General research	0	22			
	Programming	22	10			
Week 12	General research	26	29			
	Programming	0	8			
Week 13	**Status Report 1**	1	1			
	General research	15	31			
	Programming	0	3			
Week 14	General research	4	4			
	Programming	34	0			

	Week 15	General research	1	6				
		Writing report	2	3				
		Programming	34	1				
	Week 16	General research	1	19				
		Writing report	12	12				
		Programming	14	2				
	Week 17	General research	8	16				
		Writing report	22	21				
		Programming	0	0				
	Week 18	**Meeting 5**	1	1				
		Writing report	12	27				
		Programming	20	5				
	Week 19	Writing report	32	43				
		Programming	13	8				
	Week 20	**Meeting 6**	1	1				
		Writing report	32	36				
		Finalizing	8	8				

9.5 Acronyms and abbreviations

1. AB - Ada boost
2. ANN - Artificial neural network
3. API - Application programming interface
4. CART - Classification and regression tree
5. COFF - Common object file format
6. DNN - Deep neural network
7. DT - Decision tree
8. EMBER - Endgame Malware BEnchmark for Research
9. FN - False negative
10. FP - False positive
11. GBT - Gradient boosted trees
12. HGB - Histogram based gradient boosting
13. JSON - JavaScript object notation
14. kNN - K-nearest neighbor
15. LBP - Local binary patterns
16. LR - Linear regression
17. LSTM - Long short-term memory
18. ML - Machine learning
19. NB - Naive Bayes
20. NN - Neural network
21. PE - Portable executable
22. RF - Random forest
23. RNN - Recurrent neural network
24. ROC - Receiver operating characteristic
25. SVM - Support vector machine
26. TN - True negative
27. TP - True positive
28. TPR - True positive rate



NTNU

Norwegian University of
Science and Technology