Fall 12-19-2019

Hot Fusion vs Cold Fusion for Malware Detection

Snehal Bichkar
San Jose State University

Hot Fusion vs Cold Fusion for Malware Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Snehal Bichkar

Fall 2019

The Designated Project Committee Approves the Project Titled

Hot Fusion vs Cold Fusion for Malware Detection

by

Snehal Bichkar

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

Fall 2019

Dr. Mark Stamp	Department of Computer Science
Dr. Katerina Potika	Department of Computer Science
Samanvitha Basole	Department of Computer Science

ABSTRACT

Hot Fusion vs Cold Fusion for Malware Detection

by Snehal Bichkar

A fundamental problem in malware research consists of malware detection, that is, distinguishing malware samples from benign samples. This problem becomes more challenging when we consider multiple malware families. A typical approach to this multi-family detection problem is to train a machine learning model for each malware family and score each sample against all models. The resulting scores are then used for classification. We refer to this approach as "cold fusion," since we combine previously-trained models—no retraining of these base models is required when additional malware families are considered. An alternative approach is to train a single model on samples from multiple malware families. We refer to this latter approach as "hot fusion," since we must completely retrain the model whenever an additional family is included in our training set. In this research, we compare hot fusion and cold fusion—in terms of both accuracy and efficiency—as a function of the number of malware families considered. We use features based on opcodes and a variety of machine learning techniques.

ACKNOWLEDGMENTS

First, I would like to express my sincere gratitude to my advisor, Dr. Mark Stamp, for his encouragement, guidance and constant support throughout my project. His immense knowledge and valuable advice kept me motivated and helped me to concentrate on the right direction. I could not have asked for any better advisor than him.

Second, I would also like to express my gratitude to my project committee: Dr. Katerina Potika for reviewing my work and providing valuable feedback and especially Samanvitha Basole, who is also my friend, for helping me to succeed in this project.

Lastly, I would like to thank god, for giving me such loving parents and a caring husband without whom I could not have reached at this stage in my life.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

In recent years, the use of the Internet has increased dramatically, and with this increase comes increased threats to cybersecurity [2]. One common type of cybersecurity threat is malware, which is malicious software that is designed to cause harm to a computer [3]. Malware performs malicious activities such as modifying or erasing files or stealing sensitive information from a victim's computer. Different types of malware include Worms, Trojan Horses, Rogues, Password Stealers, VirTools, Trojan-Downloaders, Ransomware, and many more. With the ever-growing number and variety of malware, anti-virus products may not be efficient or effective in detecting all such threats.

Machine learning techniques have proven useful for detecting and classifying malware [4]. Several machine learning techniques, including Hidden Markov Model (HMM) [1], Support Vector Machine (SVM) [5], k-Nearest Neighbor (k-NN) [1], Random Forest [1] and deep learning [6] are used to effectively detect malware. The machine learning techniques we use for our experiments are HMM, SVM, k-NN, Random Forest, and MLP.

In this research, for malware detection, we analyze malware samples based on static features. Static features are extracted without the execution of malware executable files. Among the various static features that are available, we specifically focus on opcodes [7, 8]. We extract opcodes from a large malware dataset.

We consider ten malware families, and we form various combinations of these families to measure the effectiveness of two distinct approaches to malware detection. In the "hot fusion" approach, we train a single model, treating multiple malware families as one single

malware class. For the "cold fusion" approach, we train a model for each distinct family in our malware dataset, then use the scores generated by these models as the features for a classifier that distinguishes between malware and benign.

We perform experiments for hot fusion and cold fusion approaches in stages. In each stage, the experiments are conducted on all possible combinations of malware families. As we move to the next highest stages, one new family is included in each experiment, so that our machine learning models must deal with increasingly generic datasets. In all cases, we consider the effectiveness and efficiency of the resulting models.

This research helps us to measure the accuracy and efficiency of these two approaches for malware detection. These experiments also enable us to determine which machine learning techniques can best classify the malware and benign samples when we combine different malware families.

The remainder of this paper is organized as follows. In Chapter 2, we discuss background work and the motivation for this research. Chapter 3 explains in detail about the hot fusion and cold fusion approaches. Chapter 4 covers the dataset we use and the experimental methodology we follow. In Chapter 5, we discuss the implementation issues and the performance metrics employed. In Chapter 6, we present and analyze the results of our experiments. Finally, in Chapter 7, we summarize the paper and consider directions for future work.

CHAPTER 2

Related Work

An extensive amount of work has been done to detect and classify malware using machine learning techniques [1, 9, 10, 11, 12]. In this chapter, we discuss representative examples of this previous work and we discuss how we leverage this work.

In [13], the authors compared the generic model versus the individual model. They conducted experiments on various malware families and used different machine learning techniques such as SVM, Random Forest, k-NN, and χ^2 test by using n-gram features. According to [13], the performance of SVM on individual models was good, and they achieved 93% balanced accuracy; however, generic models resulted in a 73% balanced accuracy. Alternatively, Random Forest performed well on an individual model as well as on a generic model. There is future scope for this research in terms of using other sets of features like opcodes as well as using different machine learning techniques like HMM, which we explore in this research.

The authors in [14], propose a hybrid model, which comprises of static and dynamic features to detect malware with better accuracy. The idea is to extract both the dynamic features (like monitoring system calls and operations) and the static features (like opcodes), then form a hybrid model by combining these two feature sets. They achieved 99% accuracy using a Random Forest classifier and 98% accuracy using k-NN for $k = 10$. The hybrid model performs better than the static approach.

Opcodes are widely used for malware classification. For example, in [9], the authors used opcodes for classification. They extracted the opcodes using the IDA Pro disassembler and combined those opcodes to form one observation sequence. This observation sequence

is given as an input to train the HMM. For training purposes, only the distinct 30 high-frequency opcodes for each family were taken into consideration while they grouped all other opcodes into a single category. Then they used this trained HMM to test the samples by computing the scores of each sample. A threshold was set to determine if the file scores were above the threshold (malware) or below the threshold (benign). By performing these experiments, they successfully detected highly metamorphic virus variants with high accuracy.

In [10], the authors used HMM for malware classification. They trained multiple HMM models for several compilers and malware generators, and tested the malware samples on these trained models. The samples are scored using log-likelihood per opcode (LLPO) technique. These samples are then clustered together based on their scores. They observed that these clusters represented malware characteristics.

As we have heard of byte n-grams, the authors in [7], used opcode n-grams instead of byte n-grams to detect unknown malicious code. They performed experiments on different values of n-grams that is for unigram, bigram, trigram, four-gram, five-gram, and six-gram. For training the classifiers, they selected the top 1000 opcode n-grams having the highest document frequency values. However, the bigram opcodes outperformed the others, and Random Forest was able to achieve the highest accuracy.

The authors in [15], perform a binary classification of malware and benign samples using n-gram features. The authors experimented with twenty malware families and carried out the experiments in 20 levels. At each level, they added one family to the existing set of families and then compared the effectiveness of individual family versus multiple-family models. They used various machine learning algorithms like SVM, MLP classifier, k-NN, and Random Forest and found that k-NN performed well on generic models. They experimented with various combinations of families and concluded that as the model becomes

more generic, it becomes less effective.

In [16], the authors used opcodes as features for machine learning models. The experiments were conducted on seven malware families. In this research, the malware samples were scored using three scoring techniques, namely, HMM, OGS, and SSD. These scores were combined and were given to SVM for classification. The results showed that SVM performed well as compared to these three individual scoring techniques.

In our research, we use a similar approach for our hot fusion experiments, as described in [13, 15], where we combine multiple malware families to form a generic model and then classify the malware and benign samples. However, we choose to work on opcodes instead of n-grams, and we also use different machine learning techniques like HMM. Our cold fusion experiments are based on technique described in [16] and is discussed in Chapter 3.

CHAPTER 3

Hot Fusion and Cold Fusion Approach

In this chapter, we discuss each of these approaches that are the hot fusion and cold fusion approach in detail. These two approaches are used to form the generic models to detect malware.

3.1 Hot Fusion

As the name suggests, fusion means combining things. In the hot fusion approach, we combine multiple malware families to check if we can still detect the malware and benign samples from these generic models.

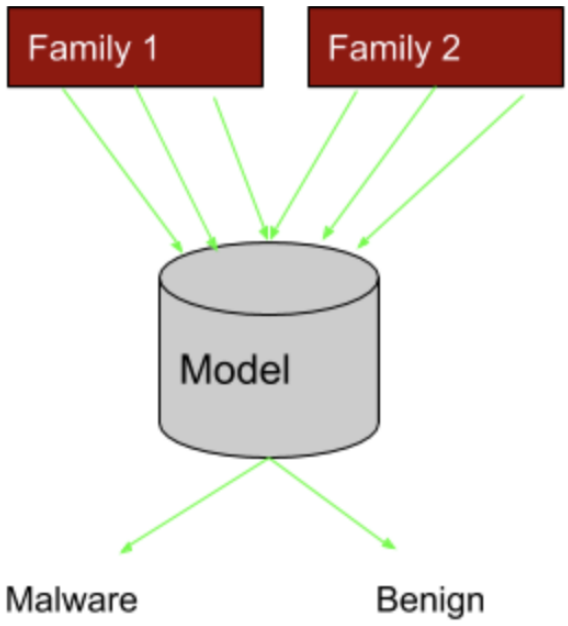

Figure 1: Hot Fusion approach

We take the samples from multiple malware families, form a model which comprises the samples from these multiple malware families, train the model and test the samples

against this model to know if the samples belong to malware set or benign set. If the new family appears, then we need to take samples from all the families, from a new model, and train and test the samples. For example, as we can see in Figure 1, we combine samples from malware family 1 and malware family 2, form a model, train this model and test the samples against this model to detect if they belong to malware or benign set. Now, if the family 3 appears, then we discard the previous model and train a new model that comprises of samples from all three malware families and then test the samples against this model. Therefore in hot fusion, we train the model every time new malware family appears.

3.2 Cold Fusion

In the cold fusion approach, we train the model for every malware family. We score

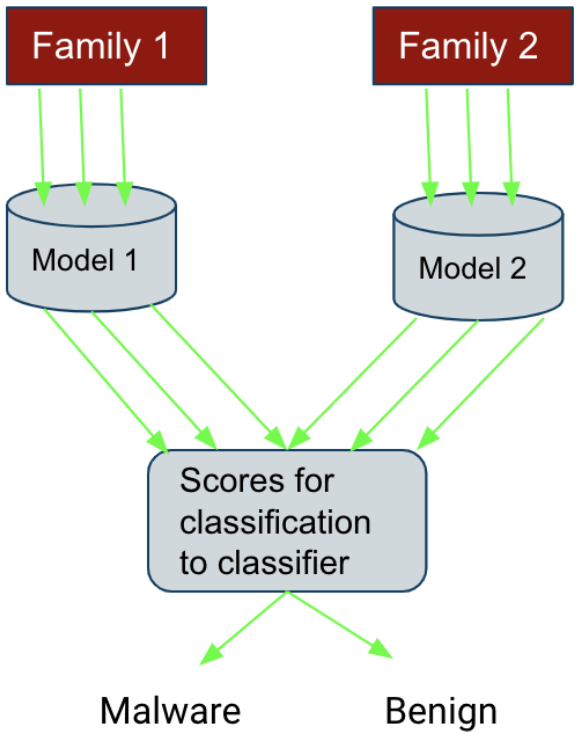

Figure 2: Cold Fusion approach

the samples against all the models. Then we combine the resultant scores of samples and give them to the classifier for the classification. Whenever a new family appears, we train a model for that family while keeping the previous models as it is. The samples are scored against all the models and are given to classifier to classify the samples as malware or benign. As we can see in Figure 2, when we have malware family 1 and malware family 2, we train the model for each malware family and then score the samples against these two models, combine the scores and give them to the classifier for classification. Now, when family 3 appears, then we only train the model for family 3 and not train the previous models. Then we score the samples against all the three models and classify the samples.

Therefore in cold fusion, we do not retrain the model whenever a new malware family appears while in hot fusion, we train the model every time new malware family appears.

CHAPTER 4

Methodology

In this chapter, we discuss the flow of our project in detail. We discuss in detail each of the modules shown in Figure 3 that is the dataset we use to conduct the experiments, the feature extraction techniques, feature selection techniques, and machine learning algorithms used in our research.

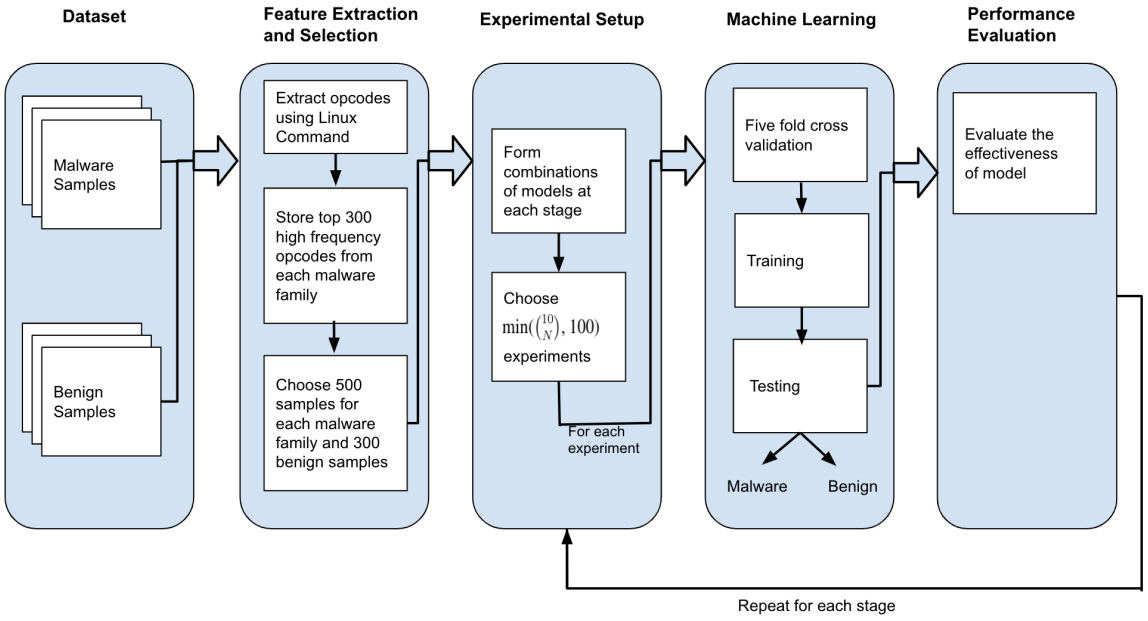

Figure 3: Project pipeline

4.1 Dataset

To run these experiments, we use a malware dataset and a benign dataset. We gather the malware dataset from VirusShare [17], which includes malware executable files, while the benign dataset includes Linux shell scripting files. The dataset in [17] consists of 500,000 malware samples. These samples belong to different families of malware, such

as Winwebsec, Zbot, Adload, and CeeInject. These malware families come from different types of malware. For example, Winwebsec belongs to Rogue, Zbot belongs to Password Stealer, and Adload belongs to Trojan-Downloader. There are 316 Linux benign samples, like chmod, cat, egrep, and bash.

4.1.1 Types of Malware Families

Our malware dataset consists of the following ten families. These families are summarized in Table 1.

Zeroaccess is a Trojan Horse that hides in the computer by using advanced rootkit. It is capable of creating a hidden file system and opening the backdoor for other malware. It downloads an application that conducts web searches and clicks on the advertisements to make money [18].

Winwebsec is a Trojan malware type. It belongs to the group of rogue security programs, which is a form of malicious software. It displays a fake virus alert on the victim's computer and makes the user buy a fake malware removal tool to remove those non-existing viruses [19].

Zbot is a Trojan Horse or a Password Stealer. It steals secret information from the computer. All credentials or bank details are jeopardized, and when it is customized through the toolkit, it can gather any information from the victim's computer. It monitors the websites that are listed in the configuration files and obtains crucial information [20].

Adload is a Trojan Downloader. It downloads an executable file and executes it, disabling the proxy settings and storing this executable file remotely. This remotely-stored

executable can be altered, or its functionality can be changed anytime to cause harm to the user's computer [21].

Renos is also a Trojan Downloader. It downloads unwanted spyware software and then displays alerts pretending the computer is infected with spyware, offering to remove the claimed spyware for a fee [22].

Vobfus is a type of Worm. It downloads other malware and infects the system, spreading via removable and network drives [23].

Vbinject is a VirTool. It hides other malware inside it and uses encryption software for shortening and obscuring its content to prevent its detection [24].

Ceeinject is also a VirTool. It hides its purpose so that the antivirus cannot detect it. This malware can have almost any purpose [25].

BHO is an Adware. It displays malicious advertisements based on the webpages visited by the user [26].

Agent is a Trojan. It downloads and installs other malware to the user's machine [27].

Table 1: Total number of malware samples for each malware family.

Family	Types	Samples
Zeroaccess	Trojan Horse	1305
Winwebsec	Rogue	4360
Zbot	Password Stealer	2136
Adload	Trojan Downloader	1226
Renos	Trojan Downloader	1568
Vobfus	Worm	1108
Vbinject	VirTool	2694
Ceeinject	VirTool	1085
BHO	Trojan	1414
Agent	Trojan	1016

4.2 Feature Extraction and Selection

For our project, we choose opcodes as a feature. In the machine language, opcodes specify the operations that need to be performed. Some of the opcodes are `mov`, `push`, `pop`, `xor`, and `sub`, as shown in Figure 4. After collecting the data, we extract opcodes

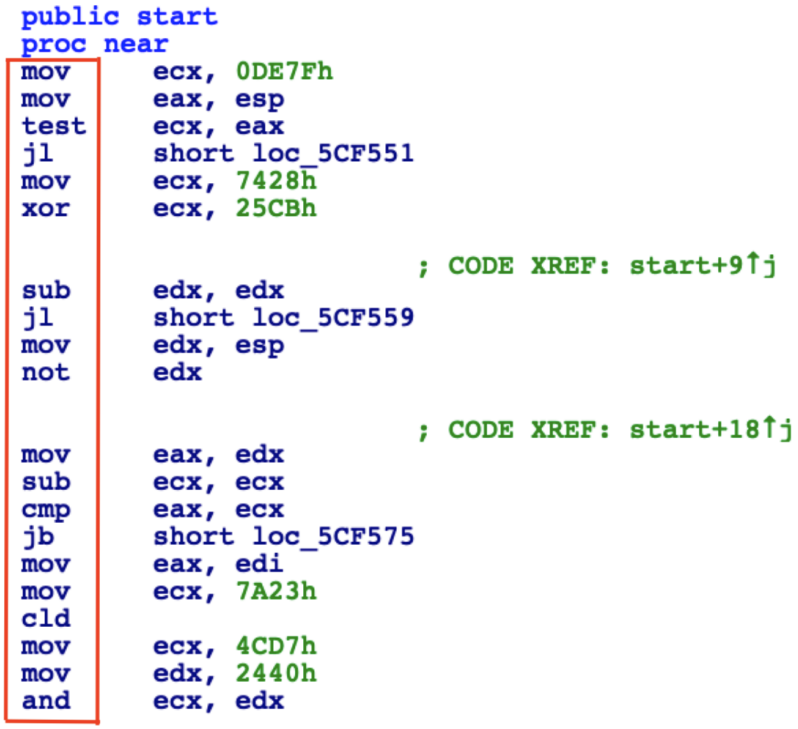

Figure 4: Disassembler view

from the malware executables and the benign Linux executables. Opcodes are extracted in two ways: first, by using IDA Pro and second by using Linux `objdump` command. Due to the high license cost of IDA Pro, we choose to extract opcodes using `objdump` command. We disassemble the malware executables and extract the opcodes, which is the highlighted part in Figure 4. We store these extracted opcodes in .txt files. Each .txt file corresponds to each sample. We eliminate incorrect opcodes by keeping only 300 distinct high-frequency opcodes from each malware family. However, there can be a n number of same opcodes

from these 300 opcodes we choose in a single malware sample. We decide to work on 500 malware samples in each malware family and 300 benign samples.

4.2.1 Experiment Setup

Each malware family has a different number of samples, as described in Table 1. To avoid biased data, we used a consistent number of samples from each malware family. For our experiments, we use 500 samples from each malware family and 300 total number of benign samples.

4.2.1.1 Hot Fusion

For the hot fusion experiments, we combine the samples from families in Table 1. For example, when we combine Zbot and Zeroaccess malware families, we choose 500 samples from Zbot and 500 samples from Zeroaccess, resulting in a total of 1000 samples from two families. We train our machine learning model on these 1000 samples and then test this model against the benign samples. Similarly, as we add more families, the total number of samples keeps increasing. So when we combine ten families, we have a total of 5000 samples.

To perform experiments, we follow the same technique used in [15]. We conduct the experiments in 10 stages. Since we are using ten malware families, we use the appropriate combinations of all ten malware families at every stage. In the first stage, we train the individual model for each malware family and compute the accuracy of each model and take the average of accuracies of these individual models. In the second stage, we combine two malware families and compute the accuracy of all the possibilities of combinations of $\binom{10}{2}$ malware families using machine learning techniques, which we explained in Section 4.2.2. To calculate the accuracy at the second stage, we take the average of accuracies of all $\binom{10}{2}$

experiments, and this average accuracy is the accuracy for the second stage. Similarly, for stage 3 we perform $\binom{10}{3}$ experiments, at stage 4 we perform $\binom{10}{4}$ experiments, for stage 5 we perform $\binom{10}{5}$ experiments, and so on until stage N where we perform $\binom{10}{N}$ experiments. So at stage 10, we have $\binom{10}{10} = 1$ possible combination and we perform only one experiment. Whenever the possible number of experiments exceed the count of hundred in each stage, we experiment with min($\binom{10}{N}$, 100) combinations, where we select the random combinations [15].

4.2.1.2 Cold Fusion

For the cold fusion experiments, the model is not retrained; instead, we train the model for a new malware family that appears. For example, first, we only have a Zbot malware family; we train the model on this family. Now, when the Zeroaccess malware family appears, then we train the Zeroaccess model individually and then score each sample against both these models and combine the scores of a sample scored against each model and then apply the machine learning technique on these combined scores to classify malware and benign samples.

We conduct experiments for cold fusion in 10 stages as we did it for the hot fusion approach. We first train all the ten families in Table 1 using a machine learning algorithm, and perform the experiments on these trained models. In the first stage, we train individual malware families, compute their accuracies, and take the average of these accuracies. In stage 2, as we did it for the hot fusion approach, we combine two malware families, so we have $\binom{10}{2}$ possible combinations of malware families, and hence we perform $\binom{10}{2}$ experiments. The accuracy at stage 2 is the average of accuracies we obtain from $\binom{10}{2}$ experiments. We perform similar experiments at stage 3 with $\binom{10}{3}$, at stage 4 with $\binom{10}{4}$, and so on until stage 10 with $\binom{10}{10}$. We perform only one experiment in stage 10, where

we combine all ten malware family models. At every stage, if the possible combinations exceed the count of 100, we only perform min($\binom{10}{N}$, 100) experiments, where we select the random combinations [15].

4.2.2 Machine Learning techniques

We conduct experiments using various machine learning algorithms. Since we split train and test data, we might lose some vital pattern in the data, so we use K-fold ($K = 5$ in our case) cross-validation to remove bias in the data. We conduct these experiments using opcodes as features for a wide variety of machine learning techniques like HMM, SVM, Random Forest, kNN, and MLP classifier.

4.2.2.1 Hidden Markov Model

HMM is a promising machine learning technique used for malware classification [9, 10, 28]. HMM is successfully implemented in the areas like genomic sequence alignment [29], speech recognition systems [30], image compression [31], data compression and pattern recognition [32]. HMM is a statistical Markov model and a discrete hill-climbing technique, and it represents the probability distributions over the series of observations [32]. It assumes the observation at the current time is generated by the previous state, which is hidden from the observer. Markov chain is a memoryless random process, and the transitions depend upon the current state and transition probability matrix. A Markov chain assumes that only current state matters to predict the future state [1, 33]. For example, we examine today's weather to predict tomorrow's weather without looking into the past [33]. HMM is based on three components: initial probabilities, observation probabilities, and state transition probabilities [1]. In HMM, the states are hidden from the observer, and the observer is only aware of the observation sequence. HMM is a doubly-embedded stochas-

tic process [30]. With the help of HMM, we can talk about observed events and hidden events [33]. According to [1], the notation for HMM is as given in Table 2.

Table 2: HMM notation

T = observation sequence length
M = distinct number of observation symbols
N = Total number of states in the HMM model
π = initial state distribution
A = matrix of state transition probability
B = matrix of observation probability
Q = distinct states of the Markov model
O = observation sequence
V = Total number of possible observations

The HMM model is denoted by λ where $\lambda = (A, B, \pi)$ and a sequence of observations O [1]. A generic view of HMM is illustrated in Figure 5

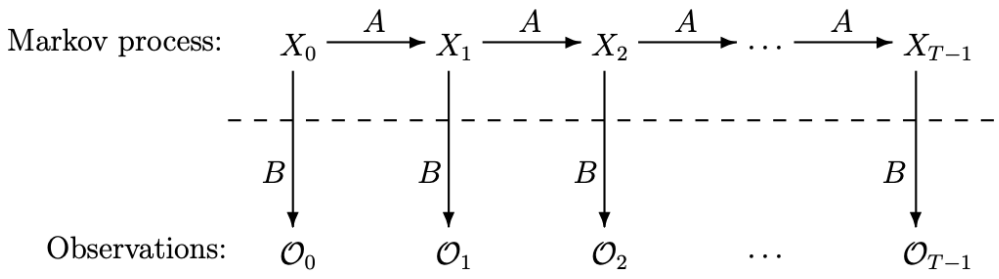

Figure 5: Hidden Markov model [1]

According to [1], to train the model, we find the parameters that best fit the observations. So, we are given the observation sequences, and from that sequence, we can determine M, where M is the unique number of characters in the observation sequence. We can select N by trial and error method, but N should be greater than or equal to 2. We have to find the model parameters A, B, and π. We find these unknown parameters by using the Baum-Welch re-estimation algorithm. Here, re-estimation is an iterative process. First, we need to initialize $\lambda = (A, B, \pi)$ with a reasonable guess or with the random values

where A, B, π must be row stochastic. We continue to re-iterate as long as the value of $P(O|\lambda)$ increases and calculate the values of α(i), β(i), γ(i,j), and γ(i). And then, we stop when $P(O|\lambda)$ does not increase by some predetermined threshold. We find $P(O|\lambda)$ after re-estimating the model. By doing this, we make sure that the model has converged. In this way, we train the model for different observation sequences.

The Forward algorithm [1] is used to score the samples. We use the state transition matrix A, observation matrix B, and initial state distribution matrix π from the above-explained training phase. We form the observation sequence for each malware sample. We compute the probability for the observation sequence that corresponds to each malware sample.

4.2.2.2 Support Vector Machine

Support Vector Machine is a supervised learning algorithm and trained on the labeled dataset. SVM is employed for binary classification, and it aims to define the optimal hyperplane that separates the classes. SVM is based on the following four main ideas [1].

- Separating hyperplane — In Figure 6, the blue pentagons, and orange pluses represent the labeled dataset. We separate these data points into two classes by drawing a hyperplane, which is a line in our case.
- Maximize margin — We can have multiple hyperplanes, as we can see in Figure 6, but only one optimal hyperplane is chosen. The optimal hyperplane is chosen by maximizing the margin between the data points in two classes to avoid bias in the classifier. In Figure 6, red line is the optimal hyperplane.
- Work in a higher-dimensional space — If the data is not linearly separable, we transform the data into higher dimensional space. It becomes easier to find optimal hy-

perplane.

- Kernel Trick — Kernels are used to project the data in higher-dimensional feature space. The results from the input space are mapped with the feature space, so we do not need to work in feature space, and it makes the computations easy. But a good choice of the kernel is important for the non-linearly separable data to be linearly separable in higher dimensions. Some of the popular kernels are the linear kernel, Gaussian radial-basis function, and polynomial learning machine.

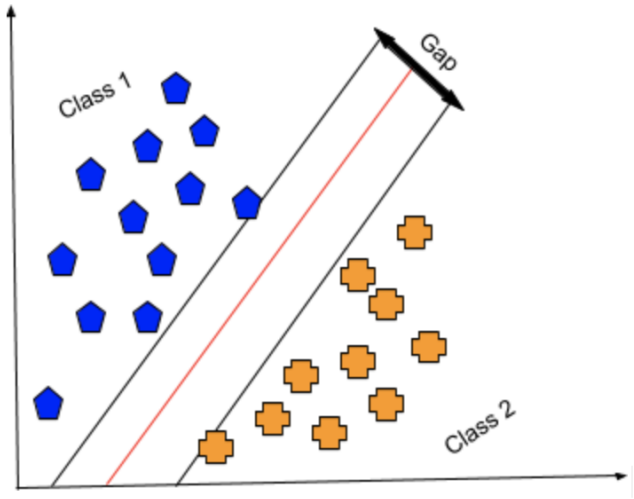

Figure 6: Separating hyperplane SVM

4.2.2.3 Random Forest

According to [34], Random Forest is an ensemble learning method and is used for classification. Random Forest is a combination of multiple decision trees. Each tree depends on the values of random vectors sampled independently and with the same distribution for all trees in the forest. It avoids the over-fitting problem that decision trees are prone to. Random Forest uses different subsets of the training data as well as different subsets of

features and hence is more robust to noise. The data is classified using a majority of votes, and the best split is chosen using a random subset.

4.2.2.4 k-Nearest Neighbors

The kNN is most easy to implement and a very simple supervised machine learning algorithm. It calculates distances from one point to all other points. Then sorts the distances, chooses the k closest distance, and then assigns the label to the point with the majority of the class vote from the k-nearest distances. In Figure 7, when we want to classify the green diamond, then we compute the distance of green diamond to all the other points. Sort the distances and then based on k-nearest neighbors (let's assume $k = 3$); we classify green diamond as red since the majority of the class vote is red [1].

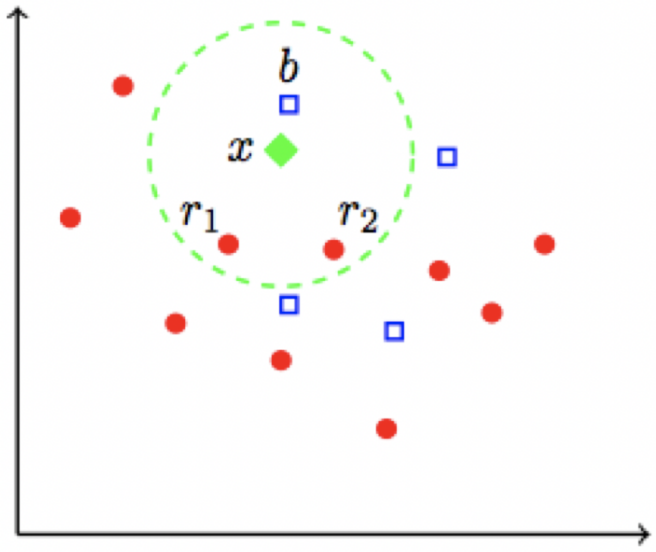

Figure 7: k-Nearest Neighbors [1]

4.2.2.5 Multi-layer Perceptron

Multi-layer Perceptron is a class of artificial neural networks. The neural networks are mainly based on neurons, weights, and activation units. MLP consists of three layers, namely, the input layer, hidden layer, output layer. MLP can have one or more hidden layers, while only one input and output layer. MLP uses backpropagation to train the model, which is a supervised learning technique. In backpropagation, the loss is feed backward in such a way that the weights are fine-tuned in order to improve the accuracy. Since the hidden layer has multiple local minima, MLP can result in different outcomes. MLP can also classify the data, which is non linearly separable [1].

4.2.3 Performance Metrics

To evaluate the effectiveness of our machine learning models, we use metrics like balanced accuracy and ROC curves. Each of these is explained in detail in Section 5.4

CHAPTER 5

Implementation and Performance Metrics

In this chapter, we discuss the environmental setup, the implementation of our project, and the performance metrics used to evaluate our experiments. We perform experiments on machine with following specifications:

- 6-Core Intel Core i7 processor
- 2.2GHz of processor speed
- 16 GB of memory
- Java and Python programming languages
- Java implementation of HMM [35]
- Scikit-Learn [36] library for SVM, Random Forest, k-NN, MLP

We disassemble each file and store the opcodes in a .txt file after extraction. We categorize the file based on the malware family labels assigned to each file and store the files under its corresponding malware family folder. Each malware family folder has 500 malware executable files and the benign folder consists of 300 Linux executables.

For our experiments, the machine learning techniques used are HMM, SVM, Random Forest, k-NN and MLP classifier for the hot fusion approach and a combination of HMM and SVM for the cold fusion approach. We also use cross-validation to solve the problem of overfitting.

5.1 HMM for Hot Fusion approach

In this section, we discuss the programming details for the training and testing of samples using HMM. At stage 1, we train the HMM model for each individual malware family and then compute the accuracy using the ROC curve. To train an HMM model in stage 2 for the hot fusion approach, we combine any two malware families. We take 500 samples from family 1 and 500 samples from family 2, and we store them in a different folder. We obtain the top twenty-nine high-frequency opcodes from the combined malware family, and the remaining opcodes are assigned the value thirty in the dictionary. We form a dictionary of opcodes mapped to a value, where the key corresponds to the top thirty high-frequency opcodes and assign the values from one through thirty. HMM works on the observation sequence, so we choose to train the HMM on 50000 observation sequence length. We use 50000 opcodes to form an observation sequence, so we divide the 50000 among the number of malware families we combined. For example, when we combine two malware families, then we use 25000 number of opcodes from family 1, and 25000 number of opcodes from family 2. We generalize this as

$$\text{Number of opcodes per family} = \frac{50000}{\text{Number of families combined}}$$

We store these opcodes in another .txt file. The opcodes from the .txt file are mapped to their corresponding value in the dictionary, and the observation sequence O is formed. We initialize the values of A, B, and π matrix randomly while keeping them row stochastic. Our project performs binary classification, which is to classify malware and benign samples, so $N = 2$, and we have 30 unique opcodes in the observation sequence, so $M = 30$. With given O, N, and M, we train the HMM to fit the observations. At each iteration, we re-estimate the model; that is, the values of A, B, and π matrices are recomputed by the HMM algorithm to fit the given observations O better. We stop this process when the model is converged, and we record the A, B, and π matrices.

For testing the samples, we use the malware samples which are not used in the training set and the benign samples. We update the A, B, and π matrix values that we obtain from the training process. Then each sample is scored using the forward algorithm (alpha pass). We score the benign samples on the same trained HMM for that particular combination of a malware family. We take the scored malware and benign samples, combine them to form a feature vector. The label vector consists of labels assigned to the corresponding malware and benign files (1 for malware samples and 0 for benign samples). Then we measure the accuracy by plotting the ROC curve. Further details about the ROC curve are explained in Section 5.4.

We repeat this process for each combination of malware families in that particular stage. For example, in stage 2, we performed $\binom{10}{2} = 45$ such experiments. So the accuracy at stage 2 is the average of the accuracies of these 45 experiments. The accuracy at each stage until N (where $N = 10$ in our case), is computed by repeating the above process.

5.2 SVM, Random Forest, *k*-NN, and MLP for Hot Fusion approach

To train the machine learning models for SVM, Random Forest, k-NN, and MLP, we first need to form a feature vector. For each malware family, we select the top thirty high-frequency opcodes.

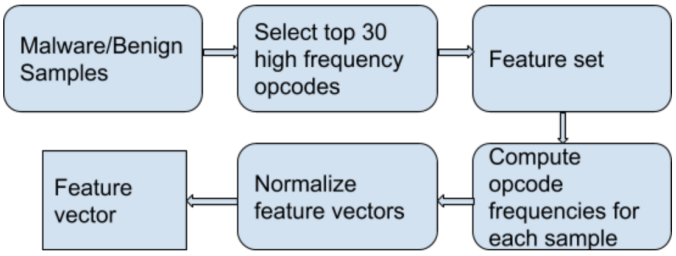

Figure 8: Opcode extraction

We compute the frequency of opcodes in each sample. To normalize the vector, we

divide these opcode frequencies by the total number of opcodes present in that sample. The flow of the feature extraction is as shown in Figure 8. This feature vector is given as an input to all the machine learning techniques in order to train the model. We use five-fold cross-validation for all machine learning techniques in order to achieve better accuracy on every chunk of data. The SVM experiments were carried using `linear` kernel. For k-NN, the value of k was set to 10. The value of parameter $n\_estimators$ of Random Forest was set to 15. And the MLP parameter solver used was `lbfgs`.

5.3 HMM-trained SVM for Cold Fusion approach

We employ SVM for our cold fusion approach, where we first train an individual malware family model using HMM, compute the scores for each sample against all the models, combine the scores, input the scores of these models to SVM for classification.

The training of individual models using HMM is carried out in the same way as we discussed in Section 5.1. Here we choose $N = 2$ as we are doing binary classification (malware and benign). Also, we choose the top 30 high-frequency opcodes and form a dictionary, so $M = 30$. We create the observation sequence O by using 50000 opcodes from a single malware family, since we are training an individual model. We provide N, M, and the observation sequence O as an input to train the HMM. The training process stops; once the model is converged, that is, we get the new values of A, B, and π matrix that better fit the given observations O. We use these new values to score the samples.

The testing process is also the same as described in Section 5.1. We score the malware and benign samples using the forward algorithm (alpha pass). Similarly, we train all the malware families individually.

To generate all possible combinations from a list of malware families, we use `combinations` function from the `itertools` module [37]. We score the samples

against all the models and combine the sample scores from all the models. For example, if we have three malware families, then we score a sample against all the three models and combines these three scores. Similarly, we do this for all the other samples. At stage 1, we calculate the accuracy of individual malware family models as we for the hot fusion approach in Section 5.1. In stage 2, we perform $\binom{10}{2} = 45$ experiments. For each experiment, we combine two malware families, score each sample from two families against the two trained HMM models, which form a feature vector. The label vector is consisting of labels assigned to the corresponding malware and benign files (1 for malware samples and 0 for benign samples). We give this feature vector as an input to SVM for classification. We split the data in the 70:30 ratio for training and testing, respectively. We also use K-Fold cross-validation ($K = 5$) with `shuffle` set to `true` and applied the SVC classifier for classification. In five-fold cross-validation, we split the entire data into five parts and train the classifier on four parts and test on the fifth part. We calculate the accuracy by using the balanced accuracy and ROC curve, as discussed in Section 5.4. We repeat this process for each combination in that stage. The accuracy at stage 2 is the average of accuracies of these 45 experiments. We repeat the above process for all stages until N (where $N = 10$ in our case) to compute the accuracy at each stage.

5.4 Performance Metrics

In this section, we discuss how we measured the effectiveness of our malware detection techniques. To quantify the effectiveness of the machine learning techniques used for malware detection, the performance metrics, we use a balanced accuracy metric and ROC curves.

5.4.1 Balanced Accuracy

Each sample is classified into one of the categories mentioned below [1];

- True Positive (TP) — positive samples correctly classified positive
- True Negative (TN) — negative samples correctly classified negative
- False Positive (FP) — negative samples incorrectly classified positive
- False Negative (FN) — positive samples incorrectly classified negative

The accuracy is the ratio of the number of samples classified correctly to the total number of samples. Accuracy is calculated as

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}. \tag{1}$$

In Equation 1, P is the total number of positive samples, and N is the total number of negative samples [1]. As we combined more malware families, the total number of samples gets doubled each time we add a new malware family. However, the benign samples remain the same. Hence to accommodate this imbalance in class, we used balanced accuracy defined as

$$\text{balanced accuracy} = \frac{1}{2}(\frac{\text{TP}}{\text{P}} + \frac{\text{TN}}{\text{N}}) \tag{2}$$

5.4.2 ROC Curve

Another way of measuring the performance of our models is through ROC curves. ROC is known as Receiver Operating Characteristic. It plots True Positive Rate (TPR) on the y-axis against False Positive Rate (FPR) on the x-axis because the threshold varies through a range of scores. It obtains the information for all possible thresholds and gives the area under the curve [1].

CHAPTER 6

Experiment Results and Analysis

In this chapter, we discuss and analyze the results we obtained using HMM, SVM, Random Forest, k-NN, and MLP machine learning techniques. We discuss the accuracy obtained by hot fusion and cold fusion approach when we make the model more generic.

6.1 Accuracies for individual malware families

In this section, we discuss the accuracies we obtained when we train individual malware family. Table 3 shows the results for individual malware family accuracy on various machine learning techniques.

Table 3: Individual malware family accuracy on various machine learning techniques

Malware Family	HMM	SVM	Random Forest	k-NN	MLP
Renos	80.00	97.29	99.61	97.22	93.59
BHO	98.00	90.97	99.11	99.36	99.11
Ceeinject	99.00	89.47	98.97	98.86	98.97
Adload	99.00	91.22	99.36	99.36	99.11
Agent	92.00	90.72	99.61	99.36	96.40
Winwebsec	97.00	98.72	99.36	99.22	99.22
Zbot	89.00	98.11	97.17	98.86	98.97
Vobfus	63.00	98.86	99.36	98.61	99.61
Zeroaccess	94.00	98.97	99.61	99.61	98.59
Vbinject	79.00	97.61	99.11	97.86	98.72

We use various machine learning algorithms as discussed in Section 4.2.2 to train the models for individual malware family and compute the accuracy of each model.

As we can see in Figure 9, HMM, which is represented by a blue bar, did not perform well on all the malware families. For example, for the malware families like Renos, Vobfus,

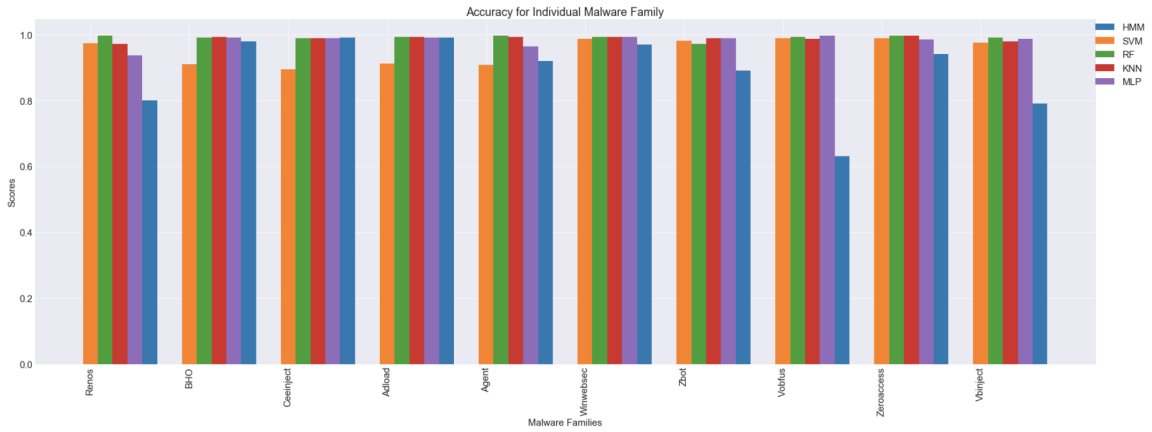

Figure 9: Accuracy for individual malware families

and Vbinject, the accuracy is below 80%. Furthermore, the lowest accuracy of 60% was scored by HMM for the Vobfus malware family. In the case of SVM, it did perform well than HMM. However, for malware families like BHO, Ceeinject, Adload, and Agent, the performance of SVM was not as good as other machine learning techniques like Random Forest and k-NN. The other techniques like Random Forest, k-NN, and MLP classifier, performed consistently well on all the malware families, and the accuracy is above 95%.

6.2 Results for Hot Fusion

In this section, we discuss the results achieved by the hot fusion approach when various machine learning techniques are employed.

6.2.1 HMM

The average accuracies at each stage for the hot fusion approach are listed in Table 4. In stage 2, we combine two malware families, and we obtain an accuracy of 81.00%. We add one more family at each stage. The average accuracy at stage 9 is 67.41%, where we combine ten malware families. As we combine malware families, the accuracy decreases. However, at stage 6, there is a small increase in accuracy as compared to stage 5. But, from

Table 4, we can say that, as the model becomes generic, it becomes less effective.

Table 4: Hot Fusion accuracies (as percentages) at each stage for HMM

Stage	Accuracy
1	89.00
2	81.00
3	79.38
4	76.42
5	74.21
6	73.06
7	73.80
8	70.54
9	68.27
10	67.41

In Figure 10, we can see that the accuracy of the combined models keeps decreasing as we go higher in stages, that is, as we add one malware family at each stage to the existing set of families.

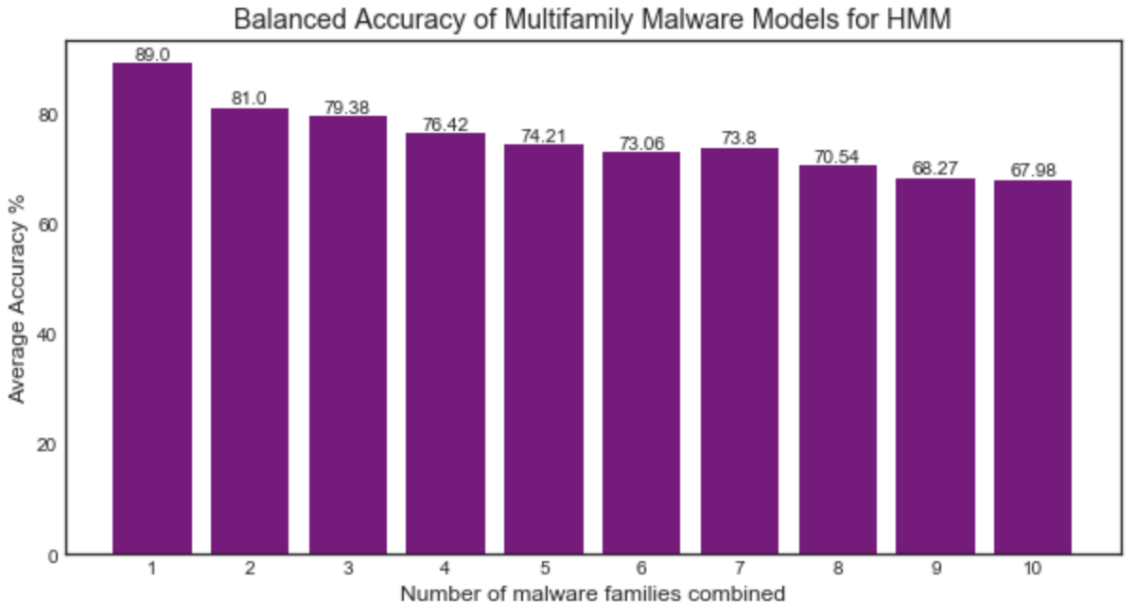

Figure 10: Average accuracies for Hot Fusion at each stage for HMM

6.2.2 SVM

Table 5 shows the accuracies of combined multiple malware family models using SVM classifier. As we can see in Figure 11, there is significant decrease in the accuracy

Table 5: Hot Fusion accuracies (as percentages) at each stage for SVM

Stage	Accuracy
1	95.10
2	94.23
3	85.00
4	76.98
5	64.68
6	57.62
7	54.36
8	52.10
9	50.19
10	50.10

of multifamily malware models when the number of malware families combined in each stage increases. The accuracy of individual malware families that is at stage 1 is 95.1%,

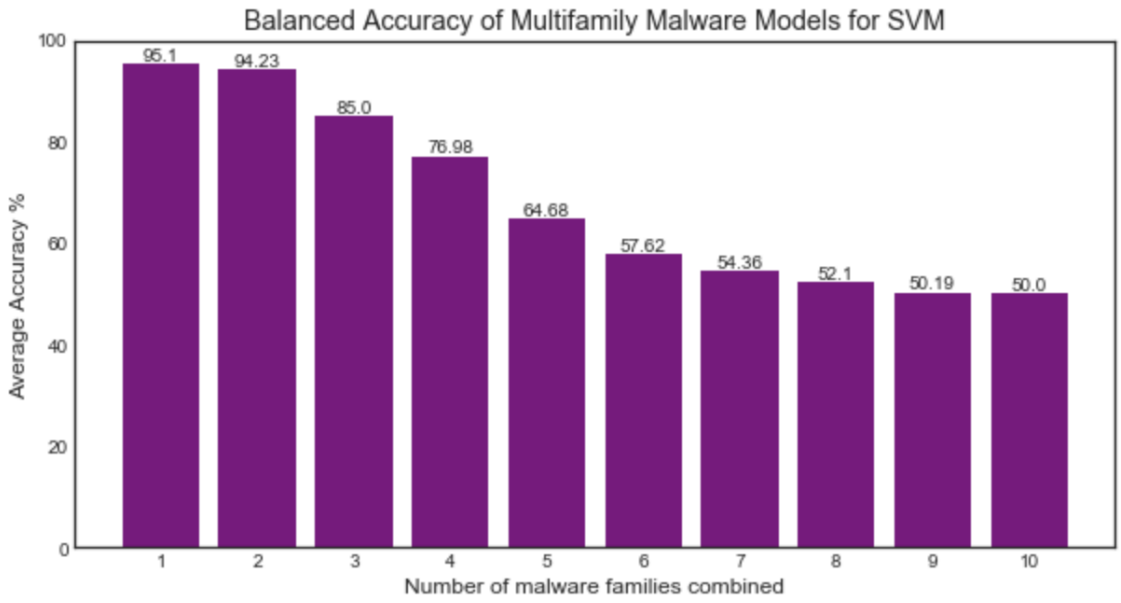

Figure 11: Average accuracies for Hot Fusion at each stage for SVM

and the accuracy at stage 2, where we combine two malware families, is 94.23%. However, the most generic model, which is at stage 10, when we combine ten malware families, the accuracy of the model drops down to 50%. Also, we can see in Figure 11, the accuracy drastically decreases from stage 2 until stage 6. Whereas, there is a constant decrease in the accuracy of generic models from stage 7.

6.2.3 Random Forest

Table 6 shows the result of the accuracies of generic models at each stage. Random Forest achieved the highest accuracy for individual models, as well as for generic models. The accuracy of an individual model is 99.12%. There is a slight decrease in the accuracy when two malware families combined. Similarly, there is a constant decrease in the accuracy of the generic models using the Random Forest classifier. The accuracy of the most generic model that is when we combine all the malware families, decreases to 93.05%. However, we can see that there is a slight increase in the accuracy at stage 6 as compared to stage 5, but the accuracy keeps decreasing further for higher stages. From Figure 12, it seems that even for the generic models, Random Forest was able to classify the malware and benign samples very well. The accuracy of all stages is above 93%.

Table 6: Hot Fusion accuracies (as percentages) at each stage for Random Forest

Stage	Accuracy
1	99.12
2	98.70
3	98.40
4	97.23
5	96.11
6	96.90
7	95.99
8	94.06
9	93.95
10	93.05

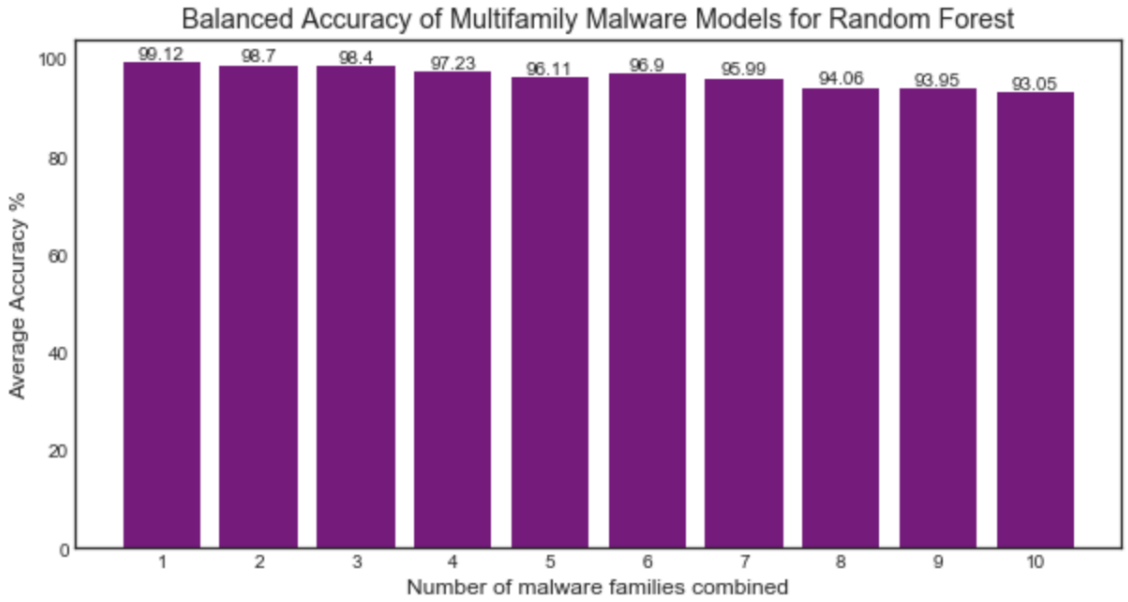

Figure 12: Average accuracies for Hot Fusion at each stage for Random Forest

6.2.4 *k*-NN

Table 7 shows results of multimalware family models using k-NN machine learning algorithm. As we can see in Figure 13, the accuracies of k-NN on generic models also shows decreasing graph. Table 7 shows the results of multimalware family models using a

Table 7: Hot Fusion accuracies (as percentages) at each stage for *k*-NN

Stage	Accuracy
1	98.83
2	97.60
3	96.65
4	96.22
5	95.60
6	94.57
7	94.59
8	93.57
9	93.26
10	92.56

k-NN machine learning algorithm. As we can see in Figure 13, the accuracies of k-NN on

generic models also show the decreasing graph.

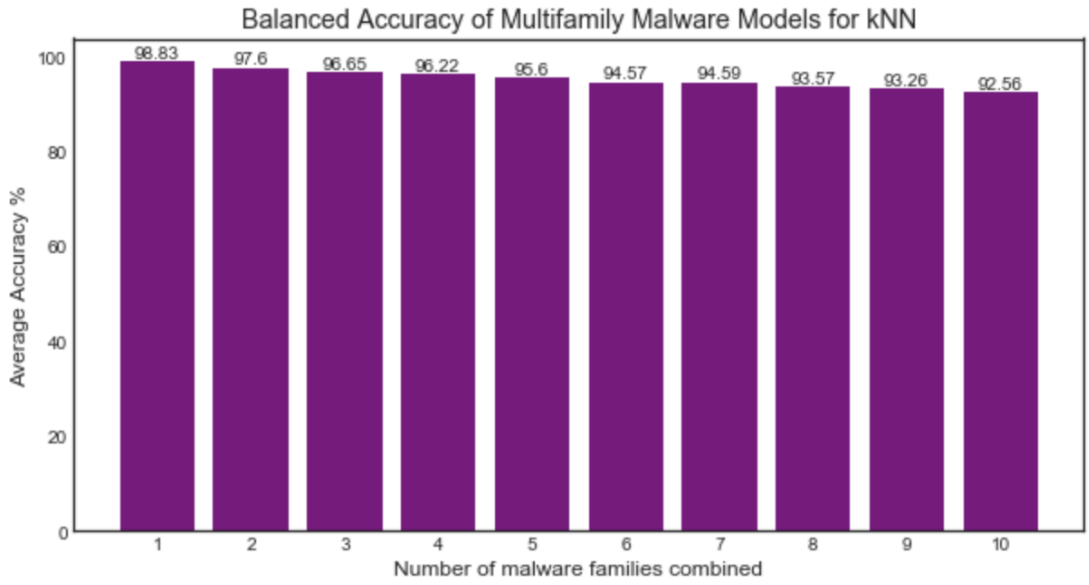

Figure 13: Average accuracies for Hot Fusion at each stage for k-NN

6.2.5 MLP

Table 8 summarizes the accuracies of the generic model in each stage wherein each stage, we add one more malware family to the existing set of malware families. The accuracy when two malware families are combined is 98.51%, and accuracy, when five malware families are combined, is 92.78%. For the later stages, the accuracy decreases by a constant value, and we can see a slight increase in the accuracy at stage 9, where nine malware families are combined, which can be due to the noise in the data. Then there was a decrease in the accuracy again at stage 10 to 89.40%. The overall trend shows a decrease in accuracy when multiple malware families are combined. This decreasing trend can be clearly seen in Figure 14. MLP did perform well than SVM and HMM; however, it did not perform as good as Random Forest and k-NN.

Table 8: Hot Fusion accuracies (as percentages) at each stage for MLP

Stage	Accuracy
1	98.22
2	98.51
3	97.68
4	94.63
5	92.78
6	92.07
7	91.05
8	90.95
9	94.56
10	89.40

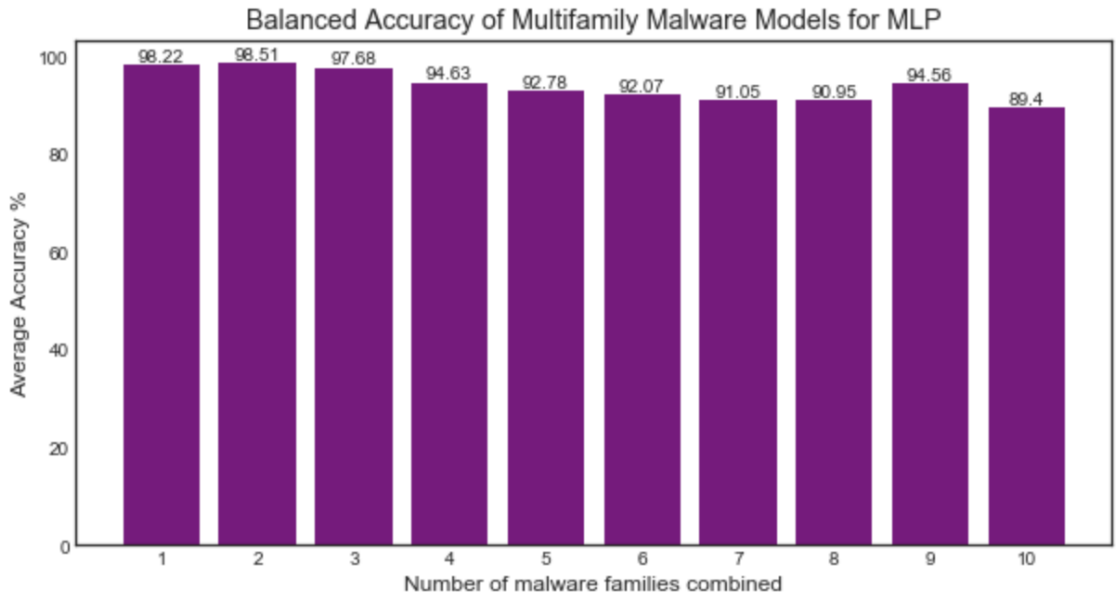

Figure 14: Average accuracies for Hot Fusion at each stage for MLP

6.2.6 Comparison of machine learning techniques

As we can see in Figure 15, HMM achieved the lowest accuracy for individual models as compared to other machine learning technologies. The graph of HMM is also decreasing as the model becomes generic. For SVM, it achieved better accuracy for individual models as compared to HMM; however, the accuracy of SVM dropped significantly as we go

higher in stages. The accuracy of SVM for a most generic model that is at stage 10 is 50.0%, which is lowest among all the other machine learning techniques used in the hot fusion approach. HMM is second from last among all the machine learning techniques

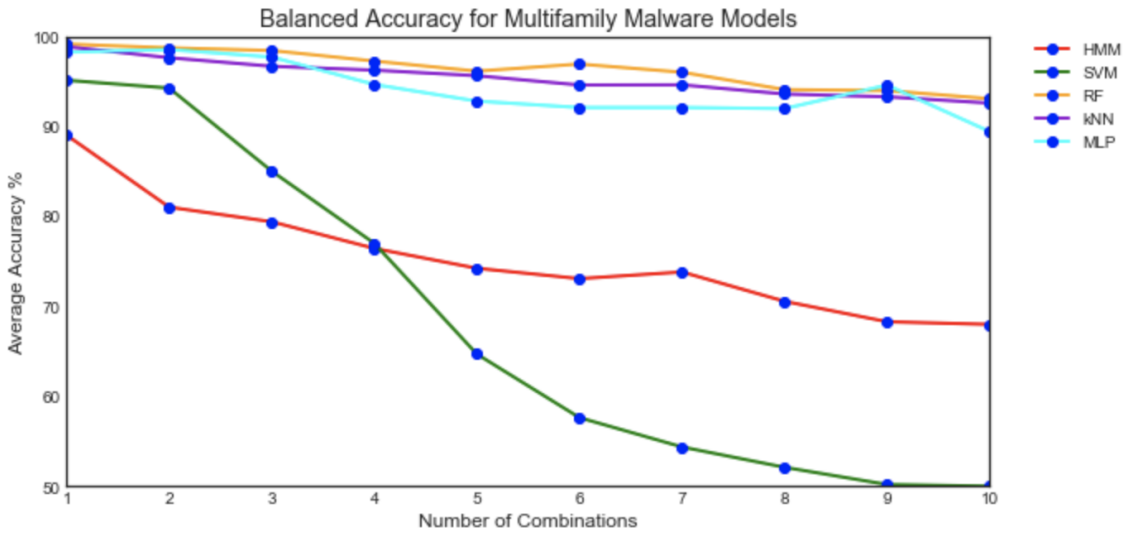

Figure 15: Balanced accuracy for multiple malware families

to achieve an accuracy of 67.41% at stage 10. Random Forest performed very well on individual and on generic models as compared to the other machine learning techniques due to the bagging technique that is used by Random Forest. Closely followed by Random Forest is k-NN supervised machine learning technique and MLP classifier, which achieved slightly less accuracy than Random Forest for generic models. This shows that it becomes challenging to detect malware when the model becomes more generic.

6.3 Results for Cold Fusion

In this section, we will discuss the results obtained for generic models using a cold fusion approach. In cold fusion, we do not retrain the previous models; instead, use the same model to score samples and only train the model for a new malware family. We discuss the results for the machine learning technique we used for the cold fusion approach.

6.3.1 HMM-trained SVM

We train the models using HMM and score samples against the trained HMM model, the scores of the samples are combined and given as an input to SVM with labels assigned as 1 and 0 for malware and benign, respectively. From Table 9, the accuracy at stage 1 with individual malware family is 89% There is a significant decrease in the accuracy of stage 2, which is 81.35%. However, the accuracy of the generic model keeps on increasing as we go higher in stages. The accuracy in stage 5 is 84.76%, where we combine five malware families, and we take an average of 100 experiments. The accuracy at stage 10 is 88%, which is higher than the accuracy in stage 2.

Table 9: Cold Fusion accuracies (as percentages) at each stage

Stage	Accuracy
1	89.00
2	81.35
3	82.70
4	84.23
5	84.76
6	85.33
7	84.90
8	87.60
9	88.90
10	88.00

In Figure 16, we can see that the accuracy of the combined models keeps increasing as we go higher in stages. When we add one malware family at each stage to the existing set of families, the classifier performs well in classifying the malware and benign samples, given the scores of each sample.

At each stage, after stage 1, there is a constant increase in accuracy in the cold fusion approach. However, we can see a negligible drop in the accuracy in stage 7, where we combined seven malware families. But the accuracy keeps increasing after that for the next

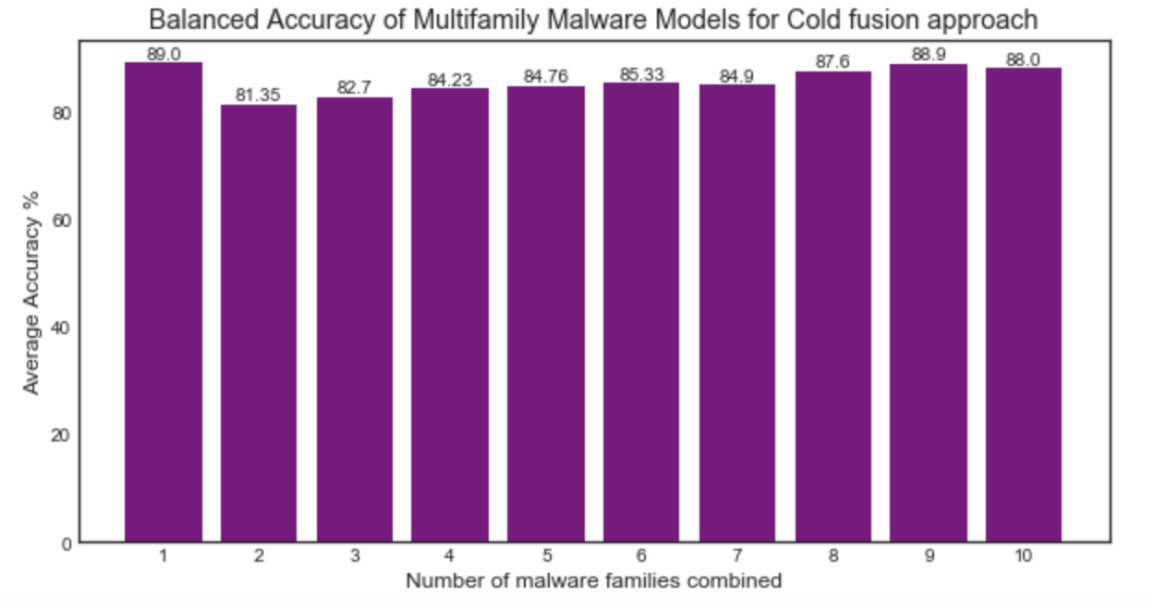

Figure 16: Average accuracies for Cold Fusion at each stage

stages.

We take the average accuracy of all the possible combinations of experiments we conduct at that particular stage. Also, at any stage, if the possible combinations are more than 100, then we perform only 100 experiments and take the average of those experiments.

6.4 Comparison of Hot Fusion and Cold Fusion

Figure 17 shows the tradeoff between the accuracy of hot fusion versus cold fusion for the generic models. Figure 17 compares the accuracies of the hot fusion approach using HMM in each stage to the accuracies of the cold fusion approach in each stage. The accuracy at stage 1 that is for the individual model is the same for the hot fusion as well as for the cold fusion. The accuracy from stage 2 keeps increasing for the cold fusion approach until stage 10, while the accuracy of hot fusion decreases as we go higher in stages. The cold fusion gets better in classifying the samples as the model becomes more generic. However, the performance of hot fusion degrades as the model becomes generic.

The accuracy of hot fusion at stage 10 is 67.14%, while the accuracy of cold fusion at stage 10 is 88%. As we can see in Figure 18, the graph of cold fusion shows an increasing trend,

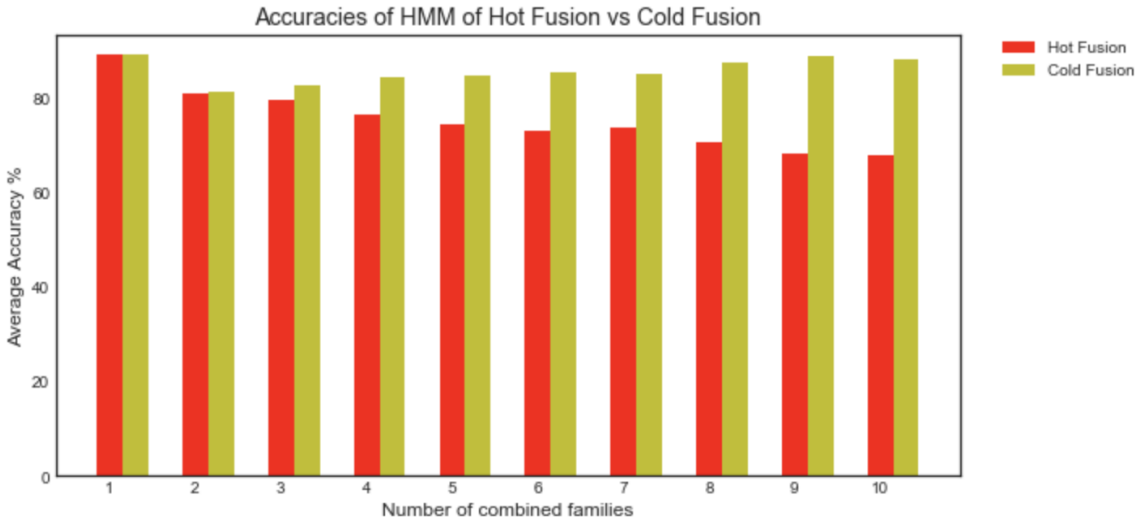

Figure 17: Accuracies of Hot Fusion (HMM) and Cold Fusion at each stage

while for hot fusion, the graph shows the decreasing trend. Besides the decreasing nature of hot fusion approach, hot fusion achieves high accuracy in each stage as compared to

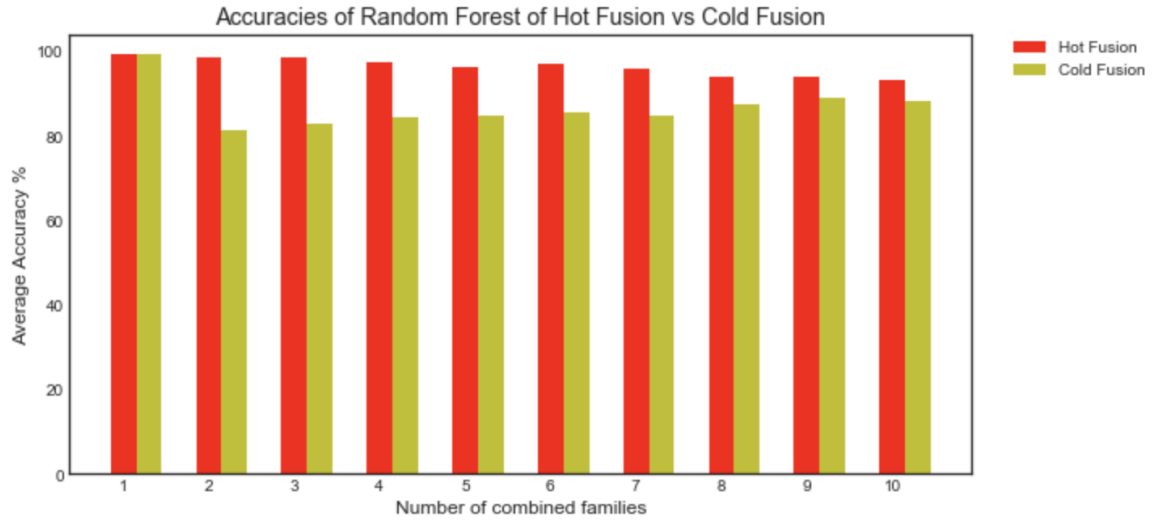

Figure 18: Accuracies of Hot Fusion (Random forest) and Cold Fusion at each stage

the cold fusion approach. We have the same accuracy for stage 1 since we train individual malware families using the same machine learning technique. Therefore the accuracy is 99.12% for both the approaches in stage 1. The accuracy for the hot fusion approach at stage 2 is 98.70%, and for cold fusion at stage 2 is 81.35%. As we can see, the accuracy of the hot fusion approach in stage 2 is much higher than cold fusion. However, this gap goes on decreasing as we go higher in stages. The Random Forest in the hot fusion approach performs well on generic models than the cold fusion approach. The k-NN and MLP techniques also performed better than the cold fusion approach for genric models.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

In this research, we conducted experiments to measure the effectiveness of hot fusion and cold fusion approaches for malware classification. For malware classification, we chose to perform the experiments using opcodes as the feature. We conducted our experiments on ten malware families. We learned about the extraction techniques of the opcodes from the malware executables. We consistently used 500 samples from each malware family to avoid bias in the data.

The experiments were performed in different stages. The experiments in stage 1 were conducted on individual malware families. In stage 2, we conducted experiments on all possible combinations by combining two malware families, which is $\binom{10}{2}$ experiments. At each stage, we kept on adding one malware family to the existing set of malware families. Therefore, at stage N, we perform $\binom{10}{N}$ experiments. Whenever the possible experiments exceeded 100 in each stage, we chose only 100 random combinations of families; that is, at each stage, we experimented with min($\binom{10}{N}$, 100) combinations, where we selected the random combinations [15].

For the experiments, the machine learning techniques used were HMM, SVM, Random Forest, k-NN, and MLP classifier. For the hot fusion approach, we used HMM for malware classification. We combined the samples from multiple malware families into one labeled set, trained HMM models on this entire set, scored the samples on the trained HMM model, and then classified the samples as malware or benign. We evaluated the effectiveness of the model using the ROC curve. The accuracy of 80.32% is achieved at stage 2

for the hot fusion approach. However, the accuracy of the generic models kept decreasing as we went higher in stages. Therefore, at stage 10, we achieved 67.41% accuracy, where we combined all the ten malware families. For the machine learning techniques like SVM, Random Forest, k-NN, and MLP, we form a feature vector where we compute the opcode frequencies in each sample and then normalized these vectors and gave them as an input to machine learning models. We see that Random Forest performed well on generic models as well as on individual models by achieving the accuracy of 99.12% for individual malware family models and 93.05% accuracy for most generic models. After Random Forest, k-NN and MLP performed well on generic models. There was a significant decrease in the accuracy of SVM as the model became more generic.

For the cold fusion approach, we trained the malware models using HMM and classified them using SVM. We trained individual models for each family, then generated a classifier based on the resulting models. Here, we did not retrain the previous models; instead, we train the model for new malware families and scored the samples against these models and then tested the samples by giving them to classifier for classification. We achieved an accuracy of 81.35% at stage 2 and the accuracy of 88% at stage 10. From the Table 9, we can see that the accuracy of the generic models kept increasing as we went higher in stages.

From the Figure 17, it seems that cold fusion performed well for the generic model as opposed to hot fusion in case of HMM technique. Besides having the increasing trend in cold fusion and decreasing trend in hot fusion, we can see from Figure 18 that the Random Forest technique of hot fusion approach performed well than the HMM- trained SVM technique of cold fusion approach. In cold fusion, the accuracy increased as we added more and more malware families, while in hot fusion, the accuracy kept decreasing.

The cold fusion approach is more efficient than the hot fusion approach, as we don't have to retrain the model every time a new malware family appears. However, in hot fusion,

we have to retrain the model every time the new malware family appears.

7.2 Future Work

In this research, we measured the effectiveness of generic models using two different ways: hot fusion and cold fusion. We observed that cold fusion performs well in case of generic models as compared to hot fusion.

Experiments could be conducted using more number of malware families and considering other features such as n-grams. Previous research [15] used more number of malware families and used n-gram features to measure the effectiveness of generic models, which is hot fusion in our case. It would be interesting to see how cold fusion performs on generic models using n-gram features.

We performed the experiments by combining multiple malware families and then classifying the malware and benign samples. We can also perform the experiments that classify the different malware families. It would be helpful if we can distinguish between different malware families so that we can take proper action against them, as we know different malware behaves differently when it affects the computer.

In this research, we considered different malware families belonging to different malware types. We can also combine the malware families that belong to the same malware type and see how the hot fusion and cold fusion approaches perform in classifying the malware of the same type. For example, we can combine malware families that belong to Trojan Horse type or families that belong to Rogue malware type and analyze the results of machine learning models to know if they can detect the malware.

For the cold fusion approach, we used HMM for training and SVM for classification. We can also use other machine learning techniques for cold fusion. For example, train the

models using HMM and classify using Random Forest classifier or train the models using HMM and classify using Convolution Neural Networks.

For the hot fusion approach, we used HMM, SVM, Random Forest, k-NN, and MLP. We can also explore other machine learning techniques like Neural Networks and clustering for the hot fusion approach. Since we are combining multiple malware families, it would be interesting to see how we can distinguish between these malware families by forming clusters of samples that belong to the same malware family.

LIST OF REFERENCES

[1] M. Stamp, *Introduction to Machine Learning with Applications in Information Security*. Chapman and Hall/CRC, 2017.

[2] Symantec, "Internet security threat report." [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

[3] Norton, "Norton security center - malware," https://us.norton.com/internetsecurity-malware.html. [Online]. Available: https://us.norton.com/internetsecurity-malware.html

[4] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[5] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[7] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, p. 1, 2012.

[8] S. R. Bragen, "Malware detection through opcode sequence analysis using machine learning," Master's thesis, Department of Computer Science and Media Technology Gjøvik University College, 2015.

[9] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[10] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden Markov models for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59–73, 2015.

[11] S. Banin and G. O. Dyrkolbotn, "Multinomial malware classification via low-level features," *Digital Investigation*, vol. 26, pp. S107–S117, 2018.

[12] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of api sequences," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014, pp. 2337–2342.

[13] N. Bagga, F. D. Troia, and M. Stamp, "On the effectiveness of generic malware models," in *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications*, ser. BASS 2017, vol. 2, 2018, pp. 442–450.

[14] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 271–280.

[15] S. Basole, M. Stamp, K. Potika, and F. Di Troia, "Multifamily malware models," 2019.

[16] T. Singh, F. Di Troia, V. A. Corrado, T. H. Austin, and M. Stamp, "Support vector machines and malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 4, pp. 203–212, 2016.

[17] S. Kim, M. Stamp, M. Moh, and F. Di Troia, "Pe header analysis for malware detection," 2018, 624.

[18] Sementec, "Trojan.zeroaccess." [Online]. Available: https://www.symantec.com/security-center/writeup/2011-071314-0410-99

[19] Microsoft, "Win32/winwebsec." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2fWinwebsec

[20] Sementec, "Trojan.zbot." [Online]. Available: https://www.symantec.com/security-center/writeup/2010-011016-3514-99?om_rssid=sr-latestthreats30days

[21] Microsoft, "Trojan.adload." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Adload&ThreatID=17567

[22] Microsoft, "Trojan.renos." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2FRenos

[23] Microsoft, "Worm.vobfus." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Vobfus.UE

[24] Microsoft, "Virtool.vbinject." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FVBInject

[25] Microsoft, "Virtool.ceeinject." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject

[26] Microsoft, "Adware.bho." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/BHO.G

[27] Microsoft, "Adware.agent." [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Agent&ThreatID=14992

[28] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Exploring hidden Markov models for virus analysis: a semantic approach," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013, pp. 5039–5048.

[29] B.-J. Yoon, "Hidden Markov models and their applications in biological sequence analysis," *Current genomics*, vol. 10, no. 6, pp. 402–415, 2009.

[30] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[31] C. Li, *Image classification and compression based on a two dimensional multiresolution hidden Markov model*. Stanford university, 1999.

[32] Z. Ghahramani, "An introduction to hidden Markov models and bayesian networks," in *Hidden Markov models: applications in computer vision*. World Scientific, 2001, pp. 9–41.

[33] Stanford, "Hidden Markov model." [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/A.pdf

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

[35] S. Mark, "Reference implementation of hmm in c (includes brown corpus)." [Online]. Available: https://www.cs.sjsu.edu/~stamp/RUA/HMM_ref.zip

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[37] "Python software foundation. itertools: Functions creating iterators for efficient looping, version 2.7," 2019-03-30. [Online]. Available: https://docs.python.org/2/library/itertools.html

APPENDIX

Additional Results

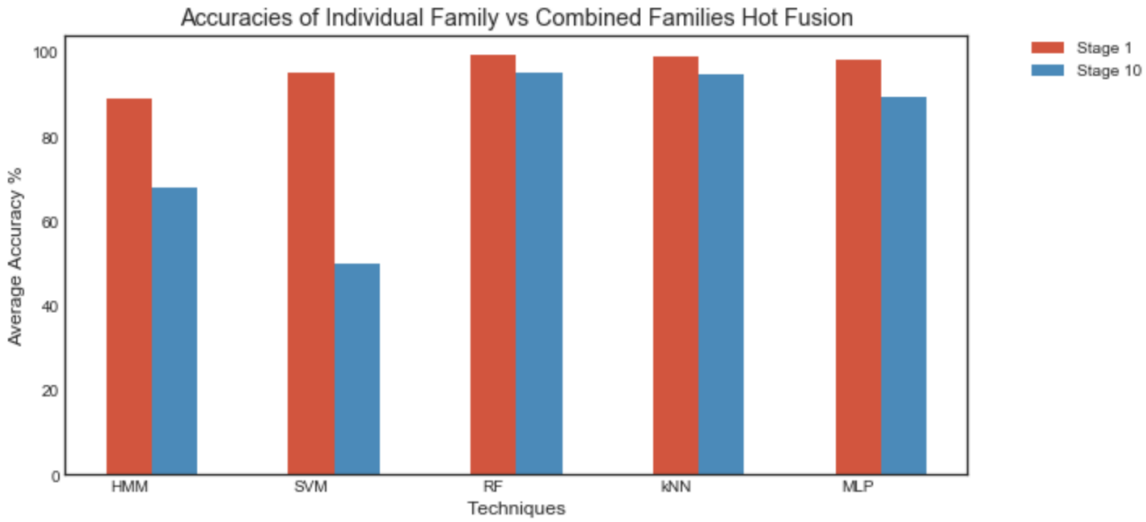

Figure A.19: Comparison of individual vs most generic model

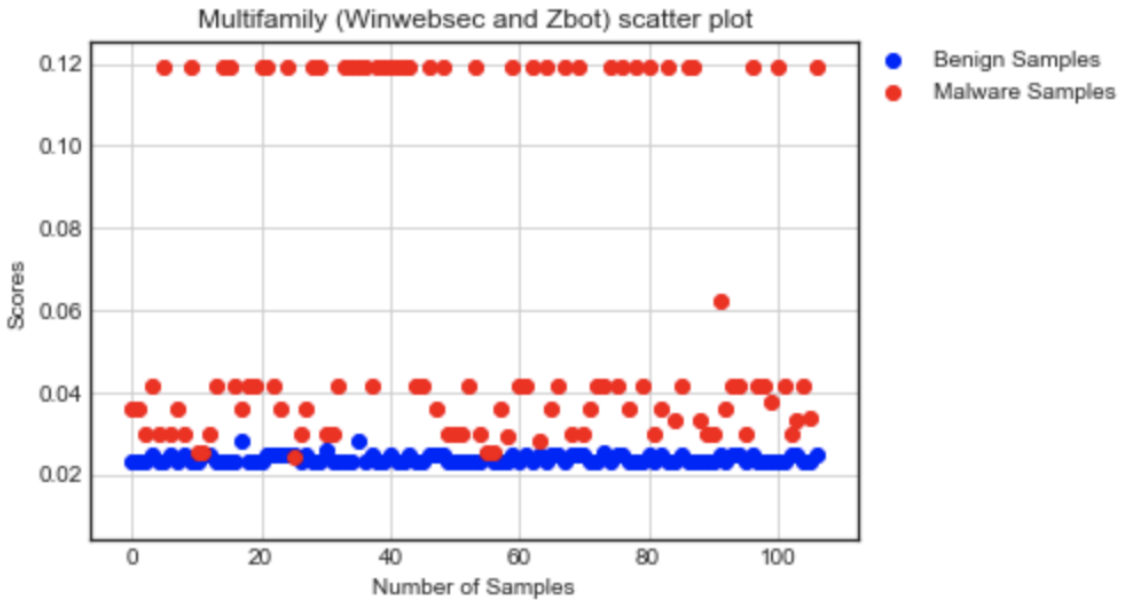

Figure A.20: Scatter plot for Winwebsec and Zbot malware families combined together using HMM

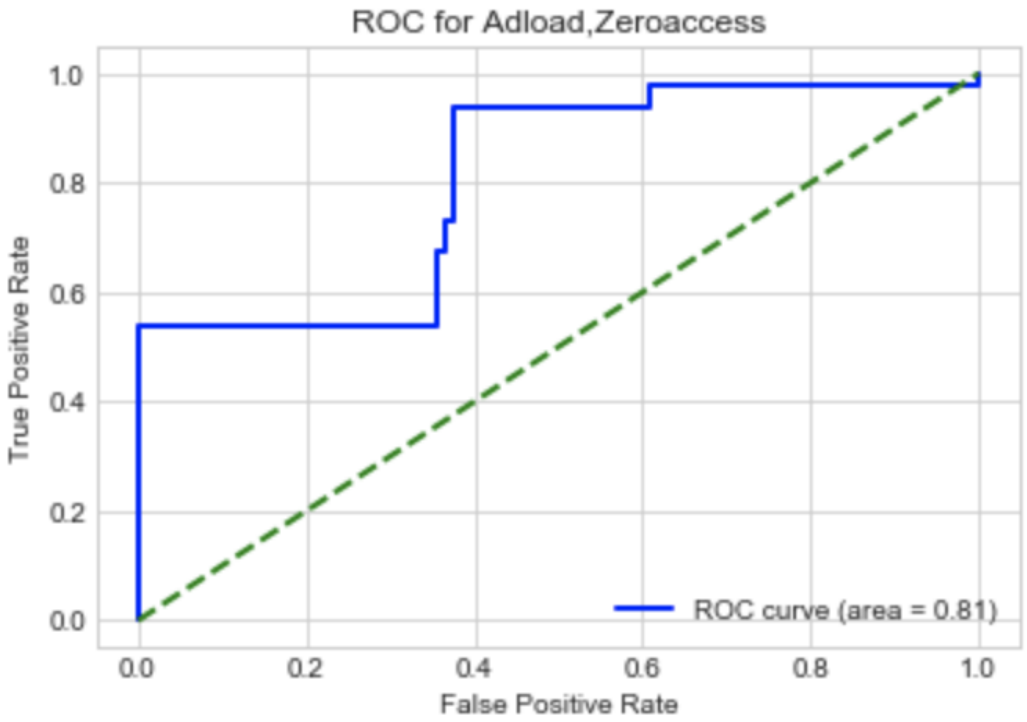

Figure A.21: ROC curve for Adload and Zeroaccess malware families combined together using HMM

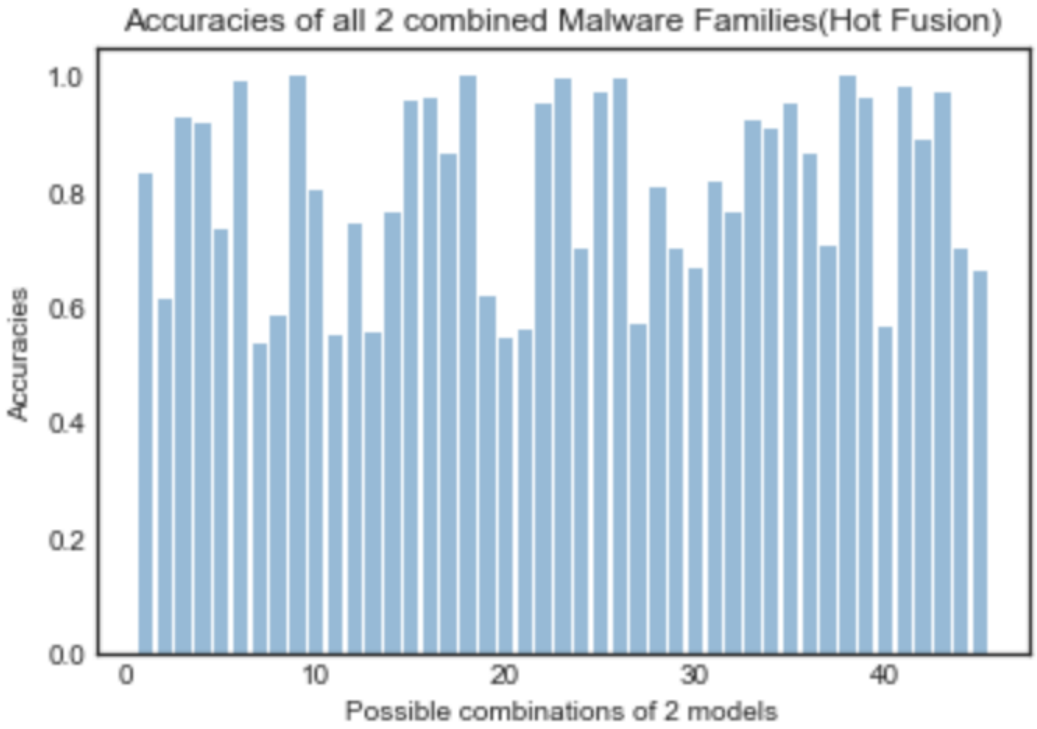

Figure A.22: Accuracies at stage 2 of combined families using HMM for Hot Fusion

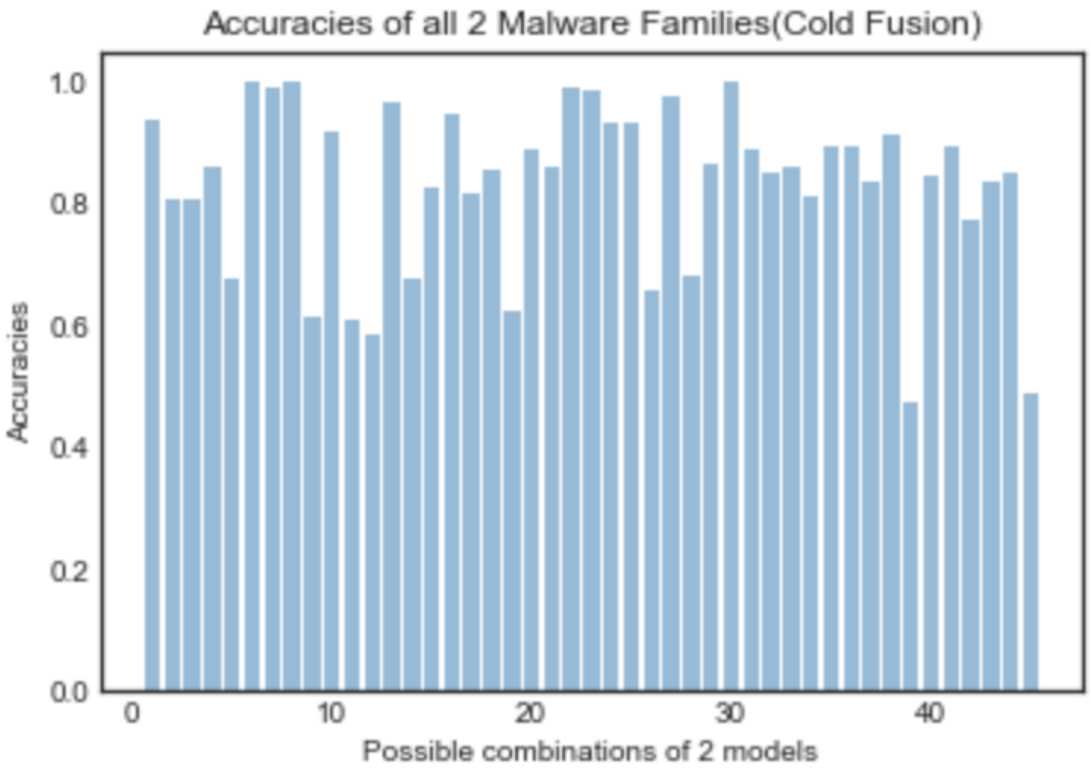

Figure A.23: Accuracies at stage 2 of combined families for Cold Fusion