Multifamily Malware Models

Samanvitha Basole
San Jose State University

Multifamily Malware Models

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

Samanvitha Basole

May 2019

The Designated Project Committee Approves the Project Titled

Multifamily Malware Models

by

Samanvitha Basole

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Mark Stamp	Department of Computer Science
Dr. Katerina Potika	Department of Computer Science
Fabio Di Troia	Department of Computer Science

ABSTRACT

Multifamily Malware Models

by Samanvitha Basole

When training a machine learning model, there is likely to be a tradeoff between the accuracy of the model and the generality of the dataset. Previous research has shown that if we train a model to detect one specific malware family, we obtain stronger results as compared to a case where we train a single model on multiple diverse families. During the detection phase, it would be more efficient to have a single model that could detect multiple families, rather than having to score each sample against multiple models. In this research, we conduct experiments to quantify the relationship between the generality of the training dataset and the accuracy of the resulting model within the context of the malware detection problem.

ACKNOWLEDGMENTS

I am truly grateful to have Dr. Mark Stamp as my advisor who has motivated and guided me through the research, patiently answered all my questions, encouraged me to tackle the challenging parts in this research, and always believed in me. His knowledge and humbleness have inspired me throughout my graduate studies.

I would like to thank my committee members, Fabio Di Troia and Dr. Katerina Potika, for their time and feedback.

I am forever indebted to my parents for always loving me, mentoring me, and supporting me through challenges. Finally, I would like to thank all my teachers and professors, especially my grandmother and my grand-uncle, for coaching me and inspiring me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Machine learning (ML) has been successfully applied in the malware domain, especially in detecting malware [1, 2]. Previous research has shown that ML approaches are better at detecting malware than traditional signature-based ones. For example, [3] and [4] have achieved high detection rates and very low false positive rates using machine learning techniques.

Machine learning can be applied to large datasets. In this paper, we consider the malware detection problem based on "big data" to quantify the effect of an increasingly generic dataset. Our goal is to examine the effectiveness of n-grams and the robustness of various machine learning techniques for detecting malware.

Our experiments are based on an extremely large and diverse malware dataset [5]. A significant part of this work involves efficiently collecting and analyzing features from a dataset that is on the order of half a terabyte.

We consider bigram features and employ a variety of machine learning techniques. Specifically, the machine learning techniques that we apply are k-nearest neighbors (k-NN), support vector machines (SVM), random forests, and deep learning [6, 7, 8, 9]. We deal with issues such as under-fitting and over-fitting by training on a large number of malware samples and by using cross-validation [8].

A complete set of experiments is conducted by first training and testing models for individual families, then we consider models designed to detect pairs of families, triples of families, and so on, up to a single model for all 20 families under consideration. In this way, we produce models that are progressively more generic [10].

We measure the accuracy of these models, which enables us to quantify the tradeoff between these important aspects of the problem. This analysis also enables us to determine which of the machine learning techniques under consideration is best able to generalize to the case where multiple diverse families are modeled.

The remainder of this paper is organized as follows. Chapter 2 discusses previous work related to this research problem. Chapter 3 outlines the methodology of our experiments. Chapter 4 describes the malware families used in the experiments, the workings of the machine learning techniques considered, and the implementation details involved in this research. In Chapter 5, we present and analyze the results of our experiments. Finally, in Chapter 6, we summarize this research and discuss possible avenues for future work.

CHAPTER 2

Background

In this chapter, we first discuss the effectiveness of using machine learning techniques for classifying malware. We also discuss the motivation and the previous work done in the domain of malware classification using n-grams, and we then give an overview of how our research compares with and builds upon the previous work.

Signature-based malware detection techniques have been very popular in detecting malware. Such techniques rely on pattern matching. Due to this, signature-based techniques are good for known malware, but as malware creators turn around to defeat such techniques and create new malware, signature-based methods cannot detect unknown malware.

In recent years, machine learning has become a successful method in detecting unknown malware with low false positives. Additionally, machine learning has been useful in detecting malware efficiently as it can scale up to handle big data.

Supervised machine learning involves training a model using known data and then validating the predictions of the model. In the malware domain, this process can be used to detect malware or to classify a sample into its malware family.

Machine learning requires features or attributes to be used in order to form the basis for classification. A few examples of features include opcodes, file headers, API calls, and n-grams.

2.1 Previous Work

An n-gram is a continuous sequence of n strings extracted from a text. In the malware domain, n-grams have been successfully and widely used as features for malware detection.

In [11], the authors use n-grams as file signatures to detect unknown malware with a low false positive ratio. They use the k-nearest neighbor algorithm to experiment with different values of n for n-grams. Their results reveal that 4-grams are best for low false positives and a maximum detection rate.

In [12], the authors achieve an accuracy of 96.64% with 4-grams and SVM. They experiment using decision tree, artificial neural network, and support vector machines to classify malware into 10 families. Their dataset consists of 12,199 files in total.

In [13], the authors describe a feature selection method using class-wise document frequency. They explain the steps in selecting relevant n-grams:

- n-grams are extracted and class-wise document frequencies are computed from virus files as well as benign files
- n-grams are sorted in descending order
- Top k n-grams are selected
- Top n-grams from virus and benign are combined to form the relevant set of n-grams

We follow a similar approach but do not consider the top n-grams from the benign dataset.

The authors in [14] claim that n-grams overestimate classification and promote overfitting. They implemented elastic-net regularized logistic regression [15] where they used a regularization path to examine the accuracy and other properties as a function of the regularized parameter. Elastic-net, a regularized method, was used with a logistic regression model. The authors achieved low weighted accuracy scores using their dataset. They used orders of magnitude more data than other n-gram-related works, but the dataset is undisclosed and comes from an industry partner which makes the study hard to replicate and compare results.

In [10], the authors performed experiments using n-grams as features for different machine learning models. Their techniques included SVM, a chi-square test, k-NN, and random forest. Their feature extraction step involved selecting the 10 most-frequent n-grams from the benign set and the malware set, resulting in a feature vector of size 20. They found that a random forest classifier is more robust than other techniques and determined the tradeoff between the generality of a model and the accuracy of its classification. They concluded that as the model becomes more generic, the accuracy goes down.

Our research uses a similar approach as described in [10] by using n-grams to determine the tradeoff between the generality and the accuracy of a model. In contrast, we use 20 families instead of 8, a different feature selection method, and different machine learning techniques, as discussed in more detail in Chapter 4.

CHAPTER 3

Methodology

In this chapter, we explain the feature extraction step, the pattern of classification experiments, and the machine learning pipeline used for each experiment. We begin by gathering a benign dataset of 1000 files and a malware dataset of 20 malware families, each family consisting of 1000 files. Chapter 4 discusses the details of this process. Once the dataset is collected, n-grams are extracted.

3.1 Feature Extraction

The top 100 bigrams from each malware file are stored as a dictionary consisting of the bigram and its frequency. Bigrams are formed by extracting bytes from the executable files. We perform the same procedure for all benign files, but we store the top 500 bigrams and their frequencies instead of the top 100. This approach of storing the dictionary worked efficiently for handling big data and extracting bigrams.

3.2 Classification Experiments

We conduct several experiments to test our hypothesis of how accuracy is affected as the model becomes more generic. Our first experiment is using five-fold cross-validation to perform a binary classification of 1000 malware files (from one *specific* family) and 1000 benign files. We perform such binary classification using 20 different malware families. This step involves detecting a specific malware family.

In the second step (or level), we combine two different malware families to form the malware class and classify files as benign or malware. At the N^{th} level, there are $\binom{20}{N}$ possible combinations of malware families, and hence, $\binom{20}{N}$ experiments. We experiment with $\min(100, \binom{20}{N})$ combinations at each step, where the combinations are selected at random. Figure 1 shows two of the many experiments at level 5. The last experiment involves combining 20 families to form the malware class, which results in a generalized dataset. Figure 2 shows the final experiment using 20 malware families.

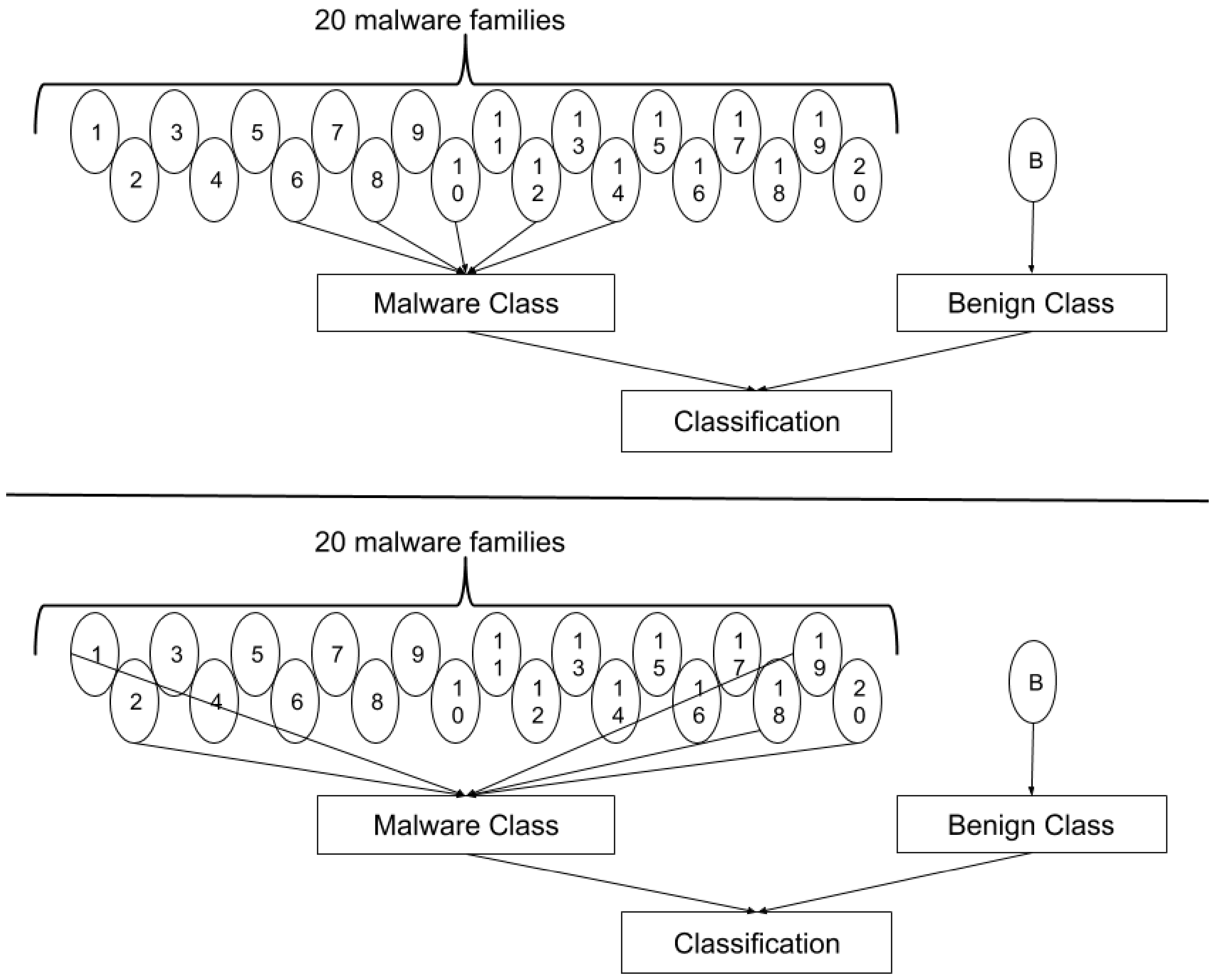


Figure 1: Two of $\binom{20}{5}$ possible experiments at level 5
The top diagram combines families 6, 8, 10, 12, and 14 into the malware class while the bottom diagram combines families 1, 2, 18, 19, and 20.

3.3 Machine Learning Pipeline

In this section, we discuss the step-by-step machine learning process for each experiment, which includes selecting a subset of features, transforming data for modeling, training and testing, and evaluating the model performance using metrics. Figure 3 illustrates the experiment pipeline consisting of the setup and the machine learning process.

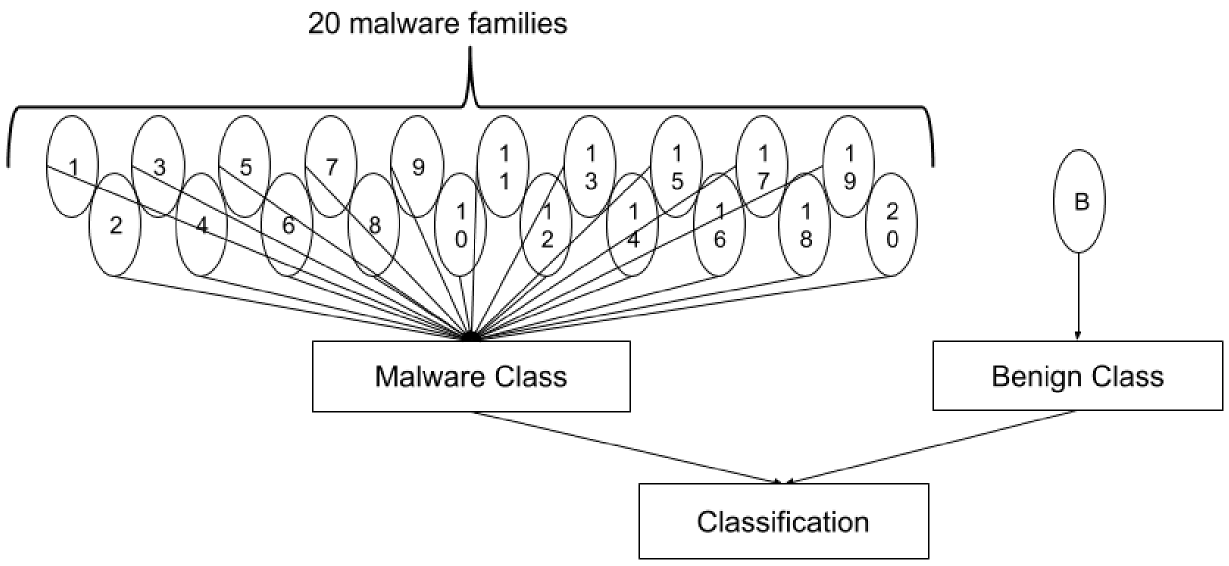


Figure 2: Experiment at level 20 (all 20 families combined as the malware class)

3.3.1 Feature Selection

From the malware class, the top 20 bigrams are selected based on their frequencies. When multiple families are combined in order to make the model generic, these multiple families belong to one class—the malware class. In this case, all families are treated equally, and the top bigrams are chosen from among these combined files.

3.3.2 Data Transformation for Modeling

Each file from the benign and the malware class is transformed into a vector of frequencies corresponding to the top 20 bigrams. We normalize the frequency vectors to make it row stochastic. These vectors form the training and the testing set. This set of vectors changes every time a family is added or removed causing the top bigrams to change. Therefore, for every experiment, we restart from the feature selection process.

3.3.3 Classification and Evaluation

For the classification step, we use cross-validation to remove any bias in accuracy caused by over-fitting. We use stratified k-fold technique where each fold maintains

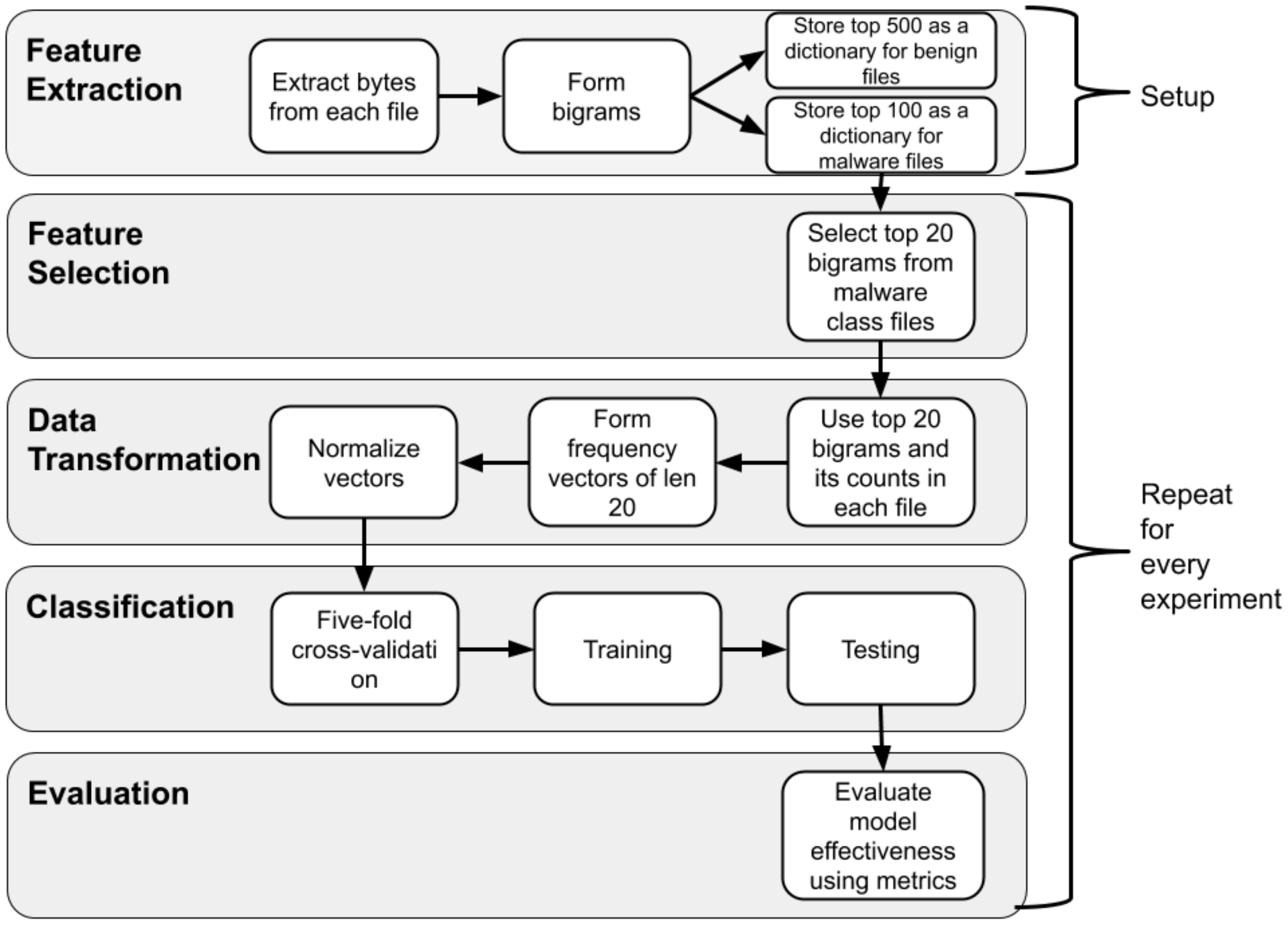

Feature Extraction
Extract bytes from each file
Form bigrams
Store top 500 as a dictionary for benign files
Store top 100 as a dictionary for malware files
Setup
Feature Selection
Select top 20 bigrams from malware class files
Data Transformation
Normalize vectors
Form frequency vectors of len 20
Use top 20 bigrams and its counts in each file
Repeat for every experiment
Classification
Five-fold cross-validati on
Training
Testing
Evaluation
Evaluate model effectiveness using metrics

Figure 3: Experiment Pipeline

the class ratio from the dataset. Five-fold cross-validation was used to split the dataset in five parts, and to train using four parts and test on one part. The final step is evaluating the results using the metrics discussed in Chapter 4

CHAPTER 4

Implementation

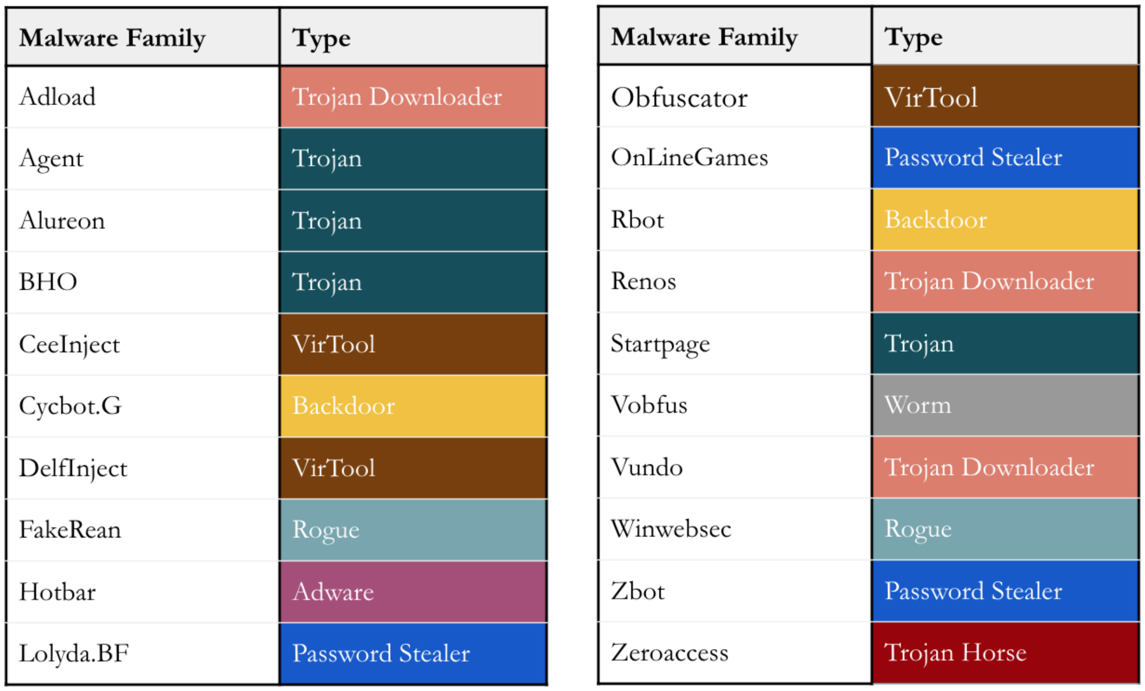
In this chapter, we give a broad summary of the malware families and the datasets used in the research. Also, we briefly discuss the machine learning techniques that were used in the experiments.

4.1 Dataset and Malware Families Used

The experiments consist of a malware dataset and a benign dataset. The benign dataset contains 1000 .exe files that came pre-installed in a Windows 10 laptop. Three malware families (Winwebsec, Zeroaccess, and Zbot) are used from the Malicia dataset [16], and the remaining seventeen families are used from a big dataset collected using VirusShare [5]. The big dataset is more than half a terabyte consisting of more than 500,000 executable files, but we used only a few families consisting of at least 1000 files. Table 1 lists the 20 families and the type of each malware family, and the family descriptions for the same type are mentioned in the same paragraph. Figure 4 shows a list of the families with a unique color label for the malware type column.

Table 1: Type of each malware family

Malware Family	**Type**	**Malware Family**	**Type**
Adload [17]	Trojan Downloader	Obfuscator [18]	VirTool
Agent [19]	Trojan	OnLineGames [20]	Password Stealer
Alureon [21]	Trojan	Rbot [22]	Backdoor
BHO [23]	Trojan	Renos	Trojan Downloader
CeeInject	VirTool	Startpage [24]	Trojan
Cycbot.G [25]	Backdoor	Vobfus [26]	Worm
DelfInject [27]	VirTool	Vundo [28]	Trojan Downloader
FakeRean [29]	Rogue	Winwebsec [30]	Rogue
Hotbar [31]	Adware	Zbot [32]	Password Stealer
Lolyda.BF [33]	Password Stealer	Zeroaccess [34]	Trojan Horse



Malware Family	Type
Adload	Trojan Downloader
Agent	Trojan
Alureon	Trojan
BHO	Trojan
CeeInject	VirTool
Cycbot.G	Backdoor
DelfInject	VirTool
FakeRean	Rogue
Hotbar	Adware
Lolyda.BF	Password Stealer

Malware Family	Type
Obfuscator	VirTool
OnLineGames	Password Stealer
Rbot	Backdoor
Renos	Trojan Downloader
Startpage	Trojan
Vobfus	Worm
Vundo	Trojan Downloader
Winwebsec	Rogue
Zbot	Password Stealer
Zeroaccess	Trojan Horse

Figure 4: Colored malware types indicate the use of generic data

Adload downloads an executable file, stores it remotely, runs the file, and disables proxy settings [17]. Renos downloads software that claims the system has a spyware and asks for a payment to remove the spyware [35]. Vundo displays pop-up ads and may download files. It uses advanced techniques to defeat detection [28].

Agent downloads trojans or other software from a remote server [19]. Alureon sends usernames, passwords, credit card data, and other confidential data from the system to hackers [21]. BHO performs actions guided by a hacker [23]. Startpage changes the browser homepage and may perform malicious activities [24].

CeeInject obfuscates to avoid being detected by the anti-virus software [36]. DelfInject sends usernames, passwords, and other personal information to hackers [27]. Obfuscator tries to obfuscate or hide itself to defeat malware detectors [18].

Cycbot.G connects to a remote server, exploits vulnerabilities, and spreads through backdoor ports [25]. Rbot gives control to hackers through a channel that allows them access to information, helps them launch attacks, and serves as a gate to spread infection [22].

FakeRean pretends to scan the system, notifies the user of issues, and asks the user to pay in order to clean the system [29]. Winwebsec runs programs that display alerts and ask the user for money to fix those issues [30].

Lolyda.BF sends information from and monitors an infected system. It can share user credentials and network activity with hackers [33]. OnLineGames steals login information of online games and tracks user keystroke activity [20]. Zbot is installed through emails, shares users' personal information with hackers, and can turn off the firewall [32].

Hotbar is an adware that shows ads on webpages and installs adware [31]. Vobfus is a worm that downloads malware and spreads through USB drives or other removable drives [26]. Zeroaccess is a trojan horse that downloads applications to click on ads to make money for malware creators [34].

4.2 Background of Classification Techniques and Metrics

In this section, we include the machine learning concepts used in the experiments. We cover k-nearest neighbors, support vector machines, random forest, and multi-layer perceptron.

4.2.1 Support Vector Machines

Support vector machines (SVM) is a supervised learning method that is based on four major ideas: generating a separating hyperplane, maximizing the margin or separation between classes, working in a higher dimensional space, and using the kernel trick. The technique uses a hyperplane to separate the labeled data into two classes. A hyperplane is one dimension less than the dimensions of the given data. Such a hyperplane tries to maximize the margin in order to generalize and reduce classification error. The algorithm works in a high-dimensional space, finds the hyperplane, and transforms the data using a kernel function [8].

For example, in Figure 5, it can be noted that there are infinite possibilities to separate the phishing and the non-phishing points. However, SVM tries to maximize the margin by selecting the bolded line.

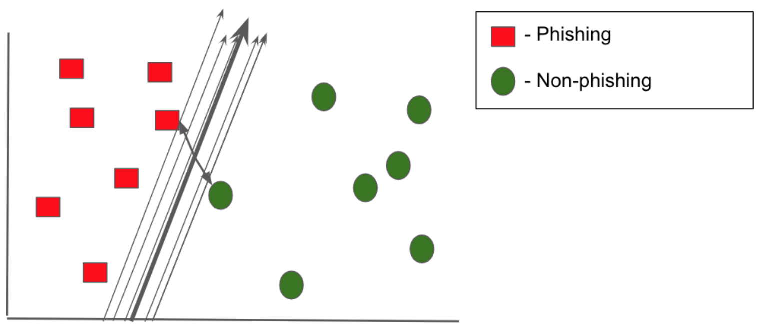


Figure 5: Example classification using SVM

4.2.2 *k*-Nearest Neighbors

One of the simplest algorithms in machine learning is k-nearest neighbors (k-NN). The training phase of k-NN consists of calculating distances from one point to all other points; while the testing phase consists of sorting the distances, choosing the k-closest distances, and then assigning the majority of class vote from the k-nearest distances. In general, training a k-NN classifier is computationally expensive for large datasets. On the other hand, classification is comparatively less complex.

Figure 6 shows an example plot of phishing and non-phishing samples. The following steps are followed to classify a point with $k = 3$.

- For $k = 3$, we choose the 3-closest neighbors as represented by the points that fall on or in the bigger circle.
- Next, we sort the distances and choose the k-nearest distances.
- We assign the majority of class vote from those 3-nearest neighbors to the test

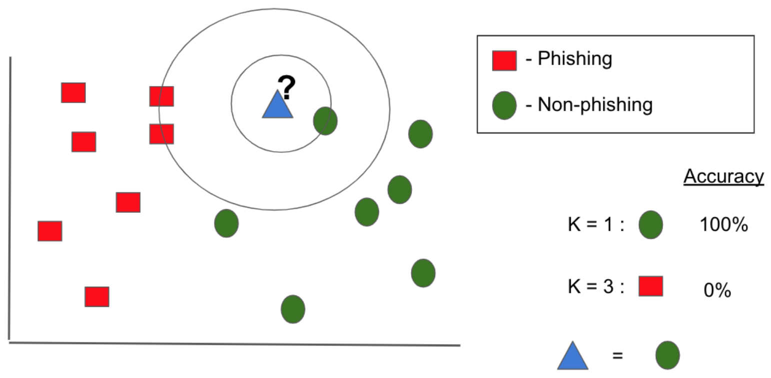


Figure 6: Example classification using k-NN

instance. In this case, we assign the test instance as a phishing sample based on the 2/3 majority vote.

4.2.3 Random Forest

A random forest generalizes a decision tree algorithm. A decision tree is a simple machine learning algorithm that is built on the idea of constructing a tree based on the features from the training data. It is easy to classify once the tree is built, but the disadvantage is that the tree tends to overfit the input data resulting in a low testing accuracy. A random forest combines multiple decision trees to generalize the training data. To do so, random forest uses different subsets of the training data as well as different subsets of features. Then, a majority vote is used to classify the data [8]. In decision trees, the best split is chosen using all variables, whereas in random forest, the best split is chosen using a random subset [37].

4.2.4 Multi-layer Perceptron

Neural networks model the brain's neurons. MLP is a type of neural network with fully-connected layers consisting of an input layer and an output layer along with one or more hidden layers. Back-propagation is used to train an MLP in order to determine its weights. The back-propagation algorithm modifies weights in such a way that errors are greatly reduced. The optimal weights are determined using partial derivatives [8].

The main advantage of this algorithm is that it can learn non-linear models using a set of features and a target. On the other hand, MLP is not robust against feature scaling and can result in different outcomes because the hidden layers have multiple local minima [38].

4.2.5 Evaluation Metrics

The final step is evaluating the model. We use balanced accuracy to evaluate the machine learning model.

4.2.5.1 Balanced Accuracy

Accuracy is defined as the number of correct classifications divided by the total number of samples and is calculated as

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

In Equation 1, TP (true positive) is the number of samples correctly classified as positive, and TN (true negative) is the number of samples correctly classified as negative. P is the total number of positive samples, and N is the total number of negative samples [8]. The positive, or malware, samples in our dataset are of size 20,000 while negative, or benign, samples are of size 1000. Due to this imbalance, we use balanced accuracy to weigh both classes equally. It is defined as

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

4.3 Programming Details

In this section, we discuss implementation-related details of our experiments. We used Python to code the experiments and a one TB hard drive to store the dataset. All experiments were run on a 2014 MacBook Pro laptop.

Each file is read as bytes, and bigrams are formed using this data. The dictionary of the top 100 bigrams and frequencies is stored in a file. The `cPickle` library [39] is used to store the dictionary representations of the .exe files. It is a module implemented in C and is used to serialize an object; that is, to store a Python object as a series of bytes. The same file can later be loaded using `cPickle`. We used this approach of storing bigrams in a file instead of opening multiple files at once and forming bigrams because forming bigrams for thousands of files is an expensive process. By using dictionaries, we are ignoring bigrams that are not very frequent. Dictionaries have the following advantages in our experiments.

- Feature selection involves combining a few dictionaries instead of extracting bytes, forming bigrams, and considering the top bigrams. We have eliminated repetitive tasks by storing that data as dictionaries.
- Creating the training and testing set involves getting the count of each of the top bigrams from each file. The time to access a value in a dictionary by its key is very fast, which helped speed up the process.

For malware files, each dictionary contained the top 100 bigrams while for benign files, each dictionary contained the top 500 bigrams. Each of the 20 malware families had a dictionary consisting of the top 1500 bigrams from that family. A `Counter` object, which is a `dict` subclass, from the `collections` module [40] was used as a dictionary.

The `combinations` function from the `itertools` module [41] generates combinations given a list of malware families.

Once all the combinations are generated, min(100, $\binom{20}{\mathrm{N}}$) combinations are chosen randomly to perform experiments at the N^{th} level. We found that running 100 experiments at each level was a good balance between accurate results and an efficient run time.

The `shuffle` function from the `random` module [42] was used to randomly select a subset of experiments to run. The function uses the Fisher-Yates shuffle which is an unbiased algorithm. The algorithm runs through the list of combinations in reverse order and randomly picks an element to exchange. After the shuffle, the first 100 combinations were chosen.

`Scikit-learn` library [38] was used to run SVM, k-NN, random forest, and MLP experiments. The `model_selection` module was used to perform 5-fold cross-validation, and the `metrics` module was used for calculating the balanced accuracy. To preserve the class ratio in experiments, the `StratifiedKFold` function was used, and to ensure that one fold would not consist of files from a specific family (especially at level 5 where 5 families are used), the `shuffle` parameter was set to `True`. A linear SVC model was used for classification using support vector machines. The neighbors for k-NN were set to 5. Random forest used 10 for the `n_estimators` and 10 for the `max_depth`. The `lbfgs` solver was used as a parameter for MLP.

CHAPTER 5

Results and Analysis

In this chapter, we discuss the results and analysis of the experiments performed on SVM, k-NN, random forest, and MLP. Although this chapter may mention accuracy, it refers to balanced accuracy. We begin by discussing the binary classification results of individual family models where we classified files from a specific family into benign and malware classes. Next, we discuss the results of multifamily models where we combined multiple malware families in the malware class and performed binary classification experiments.

5.1 Individual Family Models

In this section, we list individual accuracies, note the average balanced accuracy at level 1, and analyze the overall effectiveness of classifying one specific family as benign or malware.

5.1.1 Support Vector Machines

Table 2 shows the results of classifying individual families using SVM. The average balanced accuracy at level 1 using SVM is 88.88%. Considering that this number is an average of 20 experiments, it appears that SVM worked well to classify individual families with 1000 files in each class. Figure 7 shows an illustration created in Tableau. It can be noted that 9 families are below the average of 88% using the legend given in the figure. Two families (Agent and DelfInject) are far below the average, while three families (Vobfus, Adload, and Hotbar) are far above the average. The difference between the highest and the lowest accuracy is 19.81%. Despite the wide range, the high accuracy score implies a good performance of the families at level 1. Overall, SVM was a successful technique with a majority of families classified with a balanced accuracy greater than the average of 88.88%. A bar plot is included in the Appendix.

Table 2: SVM balanced accuracies for individual families

Malware Family	Accuracy%	Malware Family	Accuracy%
Adload	97.60	Obfuscator	86.94
Agent	79.48	OnLineGames	91.54
Alureon	89.64	Rbot	82.04
BHO	90.95	Renos	90.60
CeeInject	87.14	Startpage	86.75
Cycbot.G	92.24	Vobfus	95.15
DelfInject	77.79	Vundo	88.34
FakeRean	84.09	Winwebsec	94.05
Hotbar	97.55	Zbot	82.10
Lolyda.BF	91.80	Zeroaccess	91.60

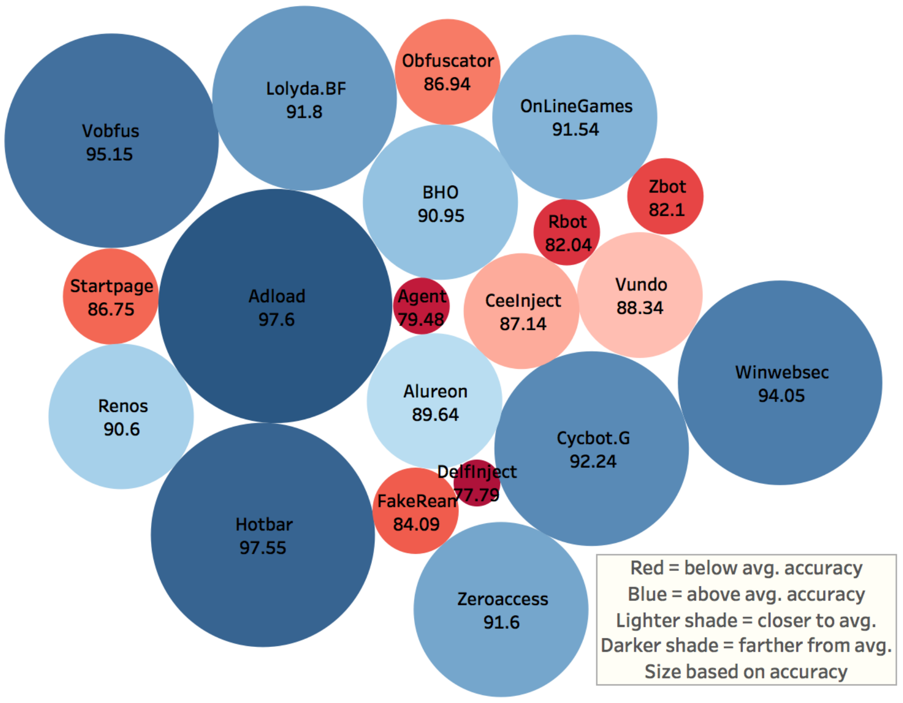


Figure 7: Individual accuracies for SVM

5.1.2 *k*-Nearest Neighbors

Table 3 shows the results of classifying individual families using *k*-NN. The average balanced accuracy at level 1 using *k*-NN is 95.87%. This is a very high average accuracy, signifying that the technique worked very well for classifying individual families. Figure 8 shows an illustration created in Tableau. It can be noted that 9 families are below the average of 95.87% using the legend given in the figure. DelfInject is far below the average while 6 families (Vobfus, Lolyda.BF, Adload, Hotbar, Zeroaccess, and Winwebsec) have accuracies greater than 98%. A bar plot is included in the Appendix. The difference between the highest and the lowest accuracy is 11.21%, which is very narrow compared to the SVM range of 19.81%. The high average accuracy implies the robustness of k-NN while the range tells us that there are a few families which are comparatively harder to detect. Overall, k-NN performed extremely well in classifying individual families with all but one accuracy greater than 90%.

Table 3: k-NN balanced accuracies for individual families

Malware Family	**Accuracy%**	**Malware Family**	**Accuracy%**
Adload	98.65	Obfuscator	91.05
Agent	94.25	OnLineGames	96.00
Alureon	94.30	Rbot	92.85
BHO	97.55	Renos	96.90
CeeInject	94.60	Startpage	95.85
Cycbot.G	98.40	Vobfus	98.10
DelfInject	87.84	Vundo	95.20
FakeRean	93.35	Winwebsec	98.85
Hotbar	98.65	Zbot	97.40
Lolyda.BF	98.60	Zeroaccess	99.05

5.1.3 Random Forest

Table 4 shows the results of classifying individual families using random forest. The average balanced accuracy at level 1 using random forest is 98.23%, which is significantly better than the results of SVM and k-NN. Figure 9 shows an illustration created in Tableau. It can be noted that 10 families are below the average accuracy

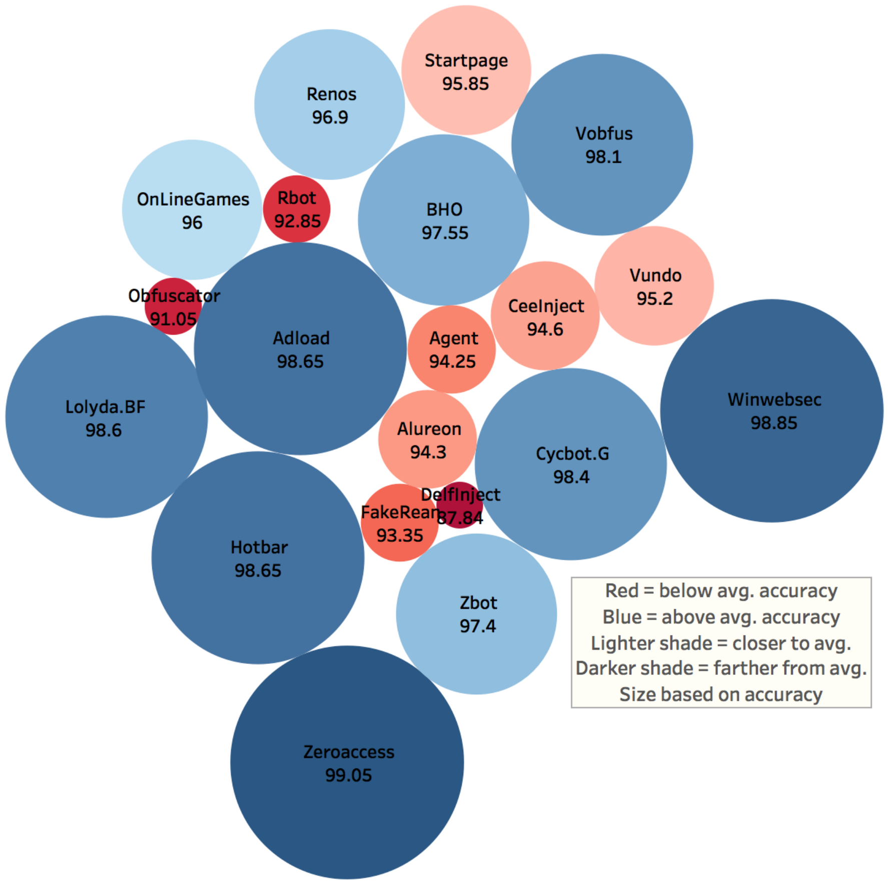


Figure 8: Individual accuracies for k-NN

of 98.23% using the legend given in the figure. DelfInject and Agent, two families with the lowest accuracies, are less than 3% below the average. Winwebsec, the family with the highest accuracy, has an accuracy of 99.95%, only 0.05% less than a perfect classification. The difference between the highest and the lowest accuracy is 4.35%. This range and the average balanced accuracy suggest that this technique produced excellent results. A bar plot showing the individual family accuracies for random forest is included in the Appendix.

Table 4: Random forest balanced accuracies for individual families

Malware Family	Accuracy%	Malware Family	Accuracy%
Adload	99.35	Obfuscator	96.10
Agent	95.90	OnLineGames	98.15
Alureon	98.00	Rbot	96.90
BHO	99.20	Renos	98.15
CeeInject	97.45	Startpage	98.75
Cycbot.G	99.35	Vobfus	99.45
DelfInject	95.60	Vundo	97.95
FakeRean	96.65	Winwebsec	99.95
Hotbar	99.35	Zbot	99.25
Lolyda.BF	99.65	Zeroaccess	99.45

5.1.4 Multi-layer Perceptron

Table 5 shows the results of classifying individual families using MLP. The average balanced accuracy at level 1 using MLP is 93.97%, which is greater than the level 1 average for SVM. A bar plot is included in the Appendix. Figure 10 shows an illustration created in Tableau. It can be noted that 10 families are below the average of 93.97% using the legend given in the figure. Alureon, DelfInject, and FakeRean have low accuracies while Adload, BHO, Hotbar, and Zeroaccess have accuracies greater than 98%. The difference between the highest and the lowest accuracy is 12.36%. This range is close to the k-NN range of 11.21%. Overall, MLP classified individual malware families resulting in a high accuracy and a narrow range.

5.1.5 Overall Results for Individual Families

Figure 11 shows boxplots for each technique. Each point on the boxplot represents the accuracy of classifying an individual family. The line on the box or the median for each boxplot indicates symmetry. It can be seen that the accuracies of SVM are left-skewed. The fact that the median of the SVM boxplot is higher than its mean of 88.88% justifies the skew. For SVM, it indicates that the high accuracies are closer

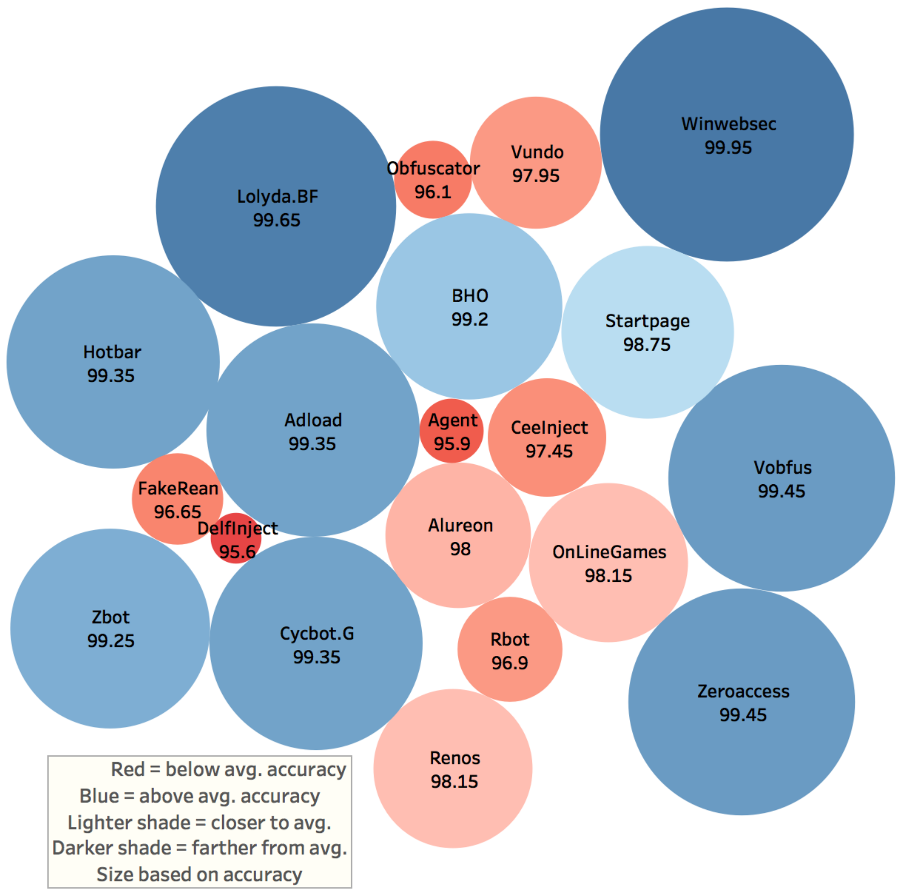


Figure 9: Individual accuracies for random forest

together than the low accuracies. The SVM box is relatively larger than the k-NN box, indicating that the accuracies of SVM are spread across a wide range while the accuracies of k-NN are concentrated in a smaller region. The short whisker at the top for the k-NN box indicates that the data points above the median are closer to each other compared to the bottom 50%. The accuracies of SVM have higher variability than the acuracies of k-NN. The box of k-NN closely resembles the box of random forest. Both have a short whisker at the top, have accuracies clustered in a few regions, and have shorter boxes compared to MLP and SVM.

MLP, like SVM, has a bigger box size with accuracies spread across the box. The

Table 5: MLP balanced accuracies for individual families

Malware Family	Accuracy%	Malware Family	Accuracy%
Adload	98.05	Obfuscator	90.95
Agent	90.65	OnLineGames	95.15
Alureon	86.74	Rbot	91.00
BHO	98.45	Renos	91.75
CeeInject	93.30	Startpage	93.60
Cycbot.G	97.80	Vobfus	97.40
DelfInject	87.65	Vundo	93.35
FakeRean	87.94	Winwebsec	97.05
Hotbar	98.70	Zbot	95.05
Lolyda.BF	95.70	Zeroaccess	99.10

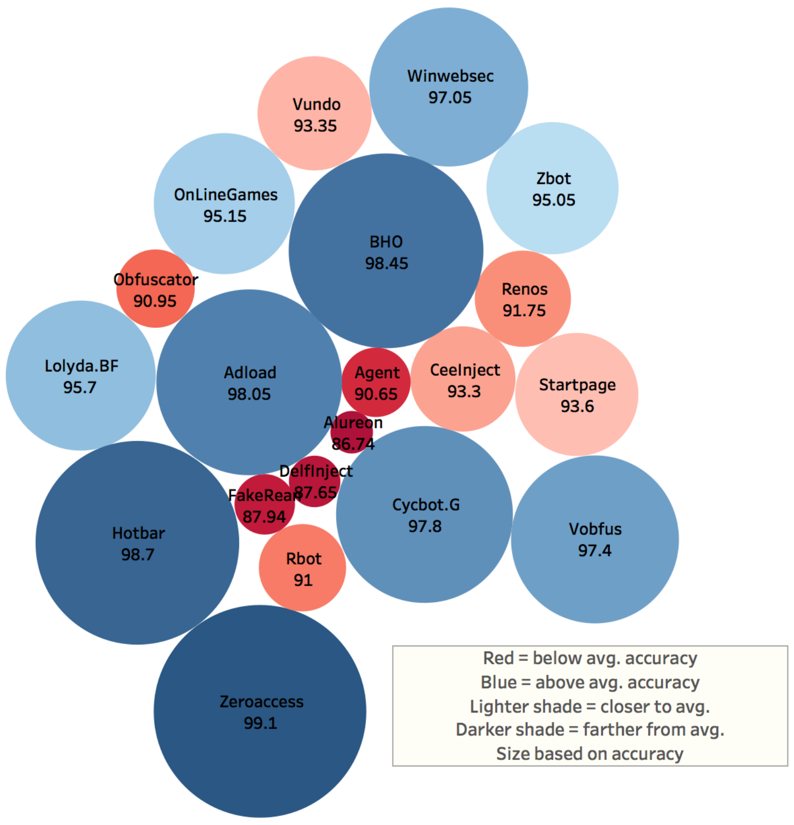


Figure 10: Individual accuracies for MLP

wide spread in the balanced accuracies of SVM shows that this technique magnified the balanced accuracies based on how hard or easy it was to detect a specific family. The length of the whiskers indicates tail length. The long whiskers of SVM indicate that the population is heavy-tailed while the k-NN data indicate that it has a light-tailed population. There is one outlier indicated by a diamond-shaped point in the k-NN data. This outlier is the balanced accuracy of DelfInject.

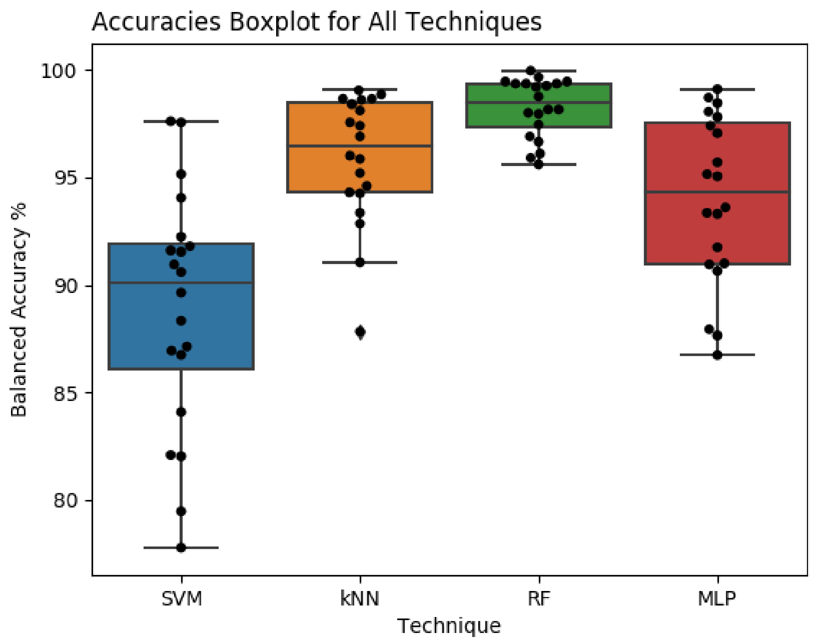


Figure 11: Individual balanced accuracies for all techniques

Figure 12 shows the ROC curves for the DelfInject family. ROC curves show the relationship between the false positive rate and the true positive rate of a classification. AUC is the area under the curve and ranges from 0.0 to 1.0 with 1.0 meaning that there are no false positives or false negatives at a certain threshold [8]. The AUC for k-NN, random forest, and MLP show excellent scores of 0.90 or higher. All four techniques are farther from the black-dotted diagonal line, indicating that all techniques have performed well on DelfInject, which was classified with the lowest accuracy for SVM,

k-NN, and random forest. It had the second lowest accuracy for MLP. The fact that all techniques had an AUC of above 0.85 shows how well all techniques performed for individual families, even for the family which was harder to detect.

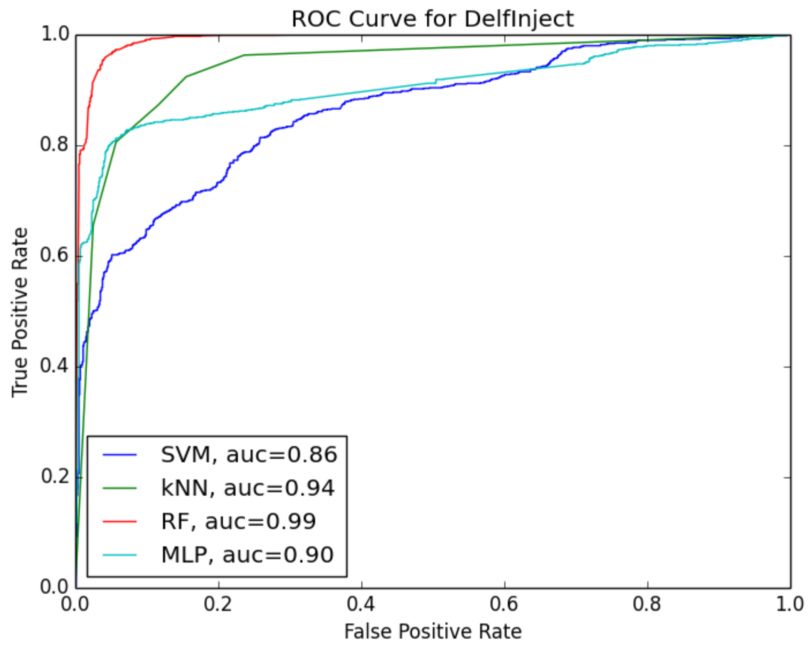


Figure 12: ROC curves for DelfInject

5.2 Multifamily Models

In this section, we list the average balanced accuracies at each level, graph the average, the lowest, and the highest accuracies, and analyze the overall effectiveness of multifamily models for SVM, random forest, k-NN, and MLP.

5.2.1 Support Vector Machines

Table 6 lists the average balanced accuracies for multifamily models using SVM. At level 1, the average balanced accuracy is 88.88%. For models with pairs of malware families, the average balanced accuracy is 78.30%. At level 20, the accuracy drops to 51.90%. Figure 13 graphs the data in Table 6 showing 3 lines which represent the average, the lowest, and the highest balanced accuracies at each level.

From level 1 to level 5, the average accuracy has drastically dropped with the maximum drop from level 1 to level 2. This means that classification experiments which consisted of two malware families in the malware class resulted in the greatest drop. From level 5 onwards, the drop is almost uniform at each level. This means that adding one more family at each level lowers the accuracy consistently.

From level 7 onwards, the three almost-parallel lines indicate that the area between the average line and the high line is almost equal to the area between the average line and the low line. In contrast, starting from level 1 to level 7, the area between the average line and the high line is greater than the area between the average line and the low line. Although the average is pulled to the low line initially, it remains centered as more families are added.

Table 6: SVM average balanced accuracies for multifamily models

Level	**Accuracy%**	**Level**	**Accuracy%**
1	88.88	11	59.21
2	78.30	12	58.47
3	72.16	13	57.85
4	67.97	14	56.38
5	65.50	15	55.41
6	63.73	16	53.82
7	62.49	17	53.38
8	61.74	18	52.98
9	61.08	19	52.42
10	60.29	20	51.90

5.2.2 *k*-Nearest Neighbors

Table 7 lists the average balanced accuracies at each level. Figure 14 shows a line graph representing the average, the lowest, and the highest balanced accuracies at each level. The overall trend shows that as the model becomes more generic with one more family added at each step, the performance is decreasing.

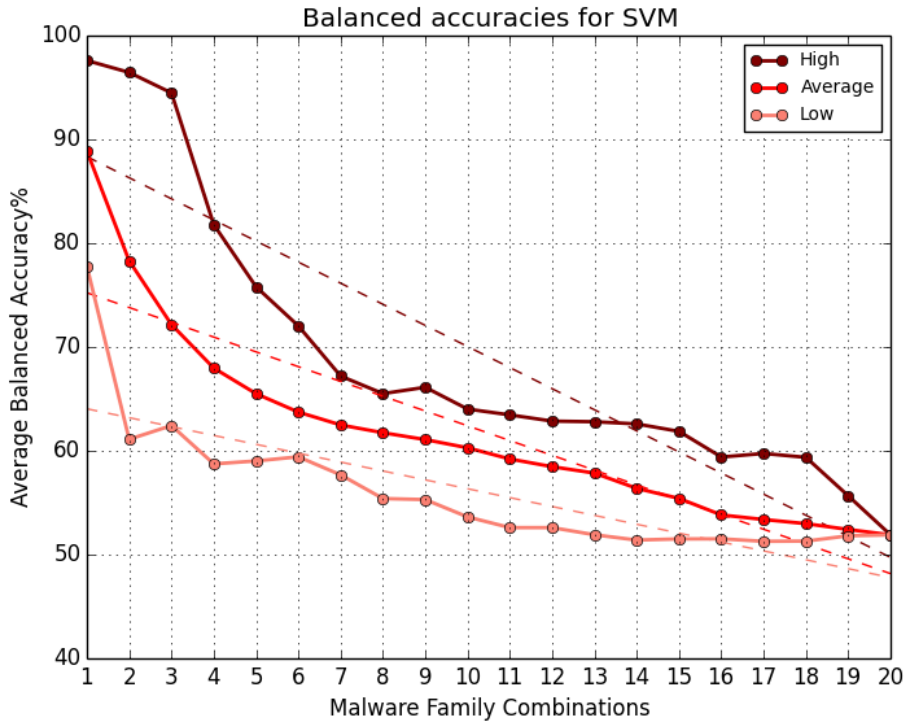


Figure 13: SVM results for all levels showing average (of 100 experiments), low, and high accuracies and linear fit lines

Table 7: k-NN average balanced accuracies for multifamily models

Level	**Accuracy%**	**Level**	**Accuracy%**
1	95.87	11	92.46
2	95.20	12	92.26
3	94.99	13	92.28
4	94.48	14	92.12
5	93.79	15	92.02
6	93.58	16	92.05
7	93.20	17	91.94
8	93.07	18	91.92
9	92.79	19	91.97
10	92.51	20	92.00

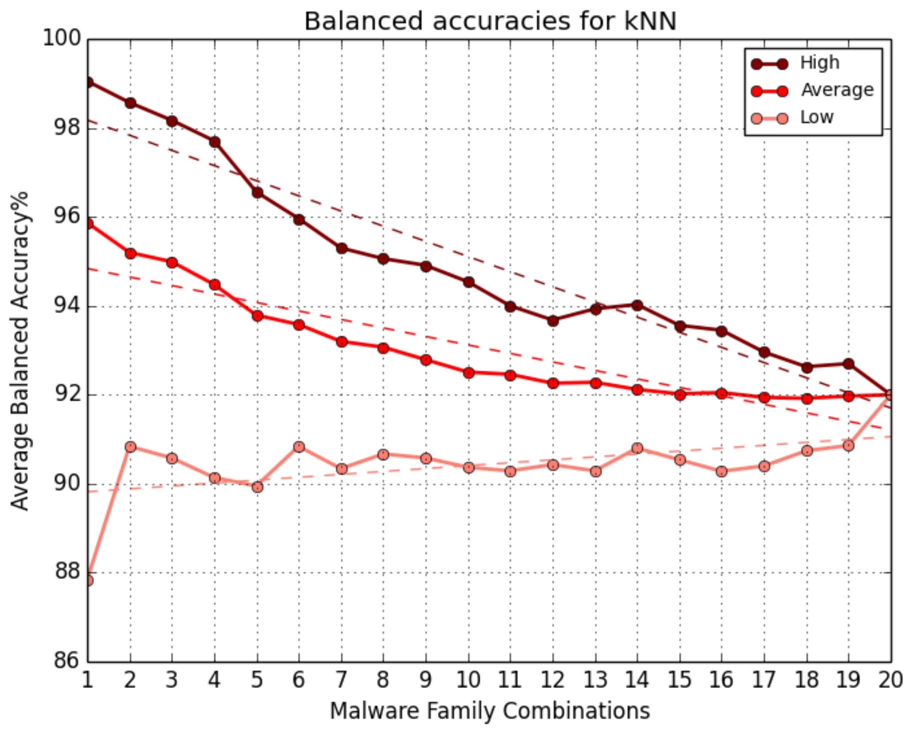


Figure 14: *k*-NN results for all levels showing average (of 100 experiments), low, and high accuracies and linear fit lines

5.2.3 Random Forest

Random forest experiments were run using different parameters for levels 1, 5, 10, 15, and 20. Table 8 shows how the balanced accuracy for random forest changes as more families are added and as the maximum depth of the tree increases, and Figure 15 graphs the data as a line chart. A parameter in random forest that specifies the maximum depth of the tree is `max_depth` [38]. At level 1, the accuracy changes from 93 to 97 to 98 for `max_depth` 2, 5, and 10 respectively. At this level, the accuracies do not differ significantly. At level 20, the accuracy changes from 50 to 83 to 93 as the `max_depth` increases from 2 to 5 to 10. It is evident from the graph that at each level, as the `max_depth` increases, the accuracy increases. Additionally, the accuracy drop at each level can be examined to conclude that a higher `max_depth` parameter of 10 is relatively robust against generic models as opposed to a parameter value of 2 or 5. It can be said that increasing the `max_depth` is preferable for generic models. For all three `max_depth` values graphed in Figure 15, the drop from level 1 to level 10 is greater than the drop from level 10 to level 20. This shows that initially, the drop is very significant. As more families are added progressively, the drop in accuracy decreases. The relatively big drop in the initial levels means that each of the 20 malware families are so different that adding one more family drops the average balanced accuracy significantly. On the other hand, when 15 families are combined, the accuracy keeps decreasing only slightly as more families are added because the model is already generic enough that adding one more family does not make a huge impact.

Table 8: Random forest balanced accuracies for changing depths

Level	**max_depth=2**	**max_depth=5**	**max_depth=10**
1	92.85	97.46	98.23
5	70.01	92.34	96.77
10	56.78	86.10	94.81
15	53.98	82.90	93.35
20	50.00	82.53	92.87

Table 9 shows the average balanced accuracies for multifamily models using random forest. The average balanced accuracy for classifying a specific family as malware or benign is 98.23%. When two families are combined, the average balanced accuracy drops to 97.94%. Finally, when all 20 families are combined in the malware class, the accuracy drops to 92.87%. Figure 16 graphs the data in Table 9 and shows three lines representing the average, the lowest, and the highest balanced accuracies at each level. The average is taken using 100 randomly-chosen experiments.

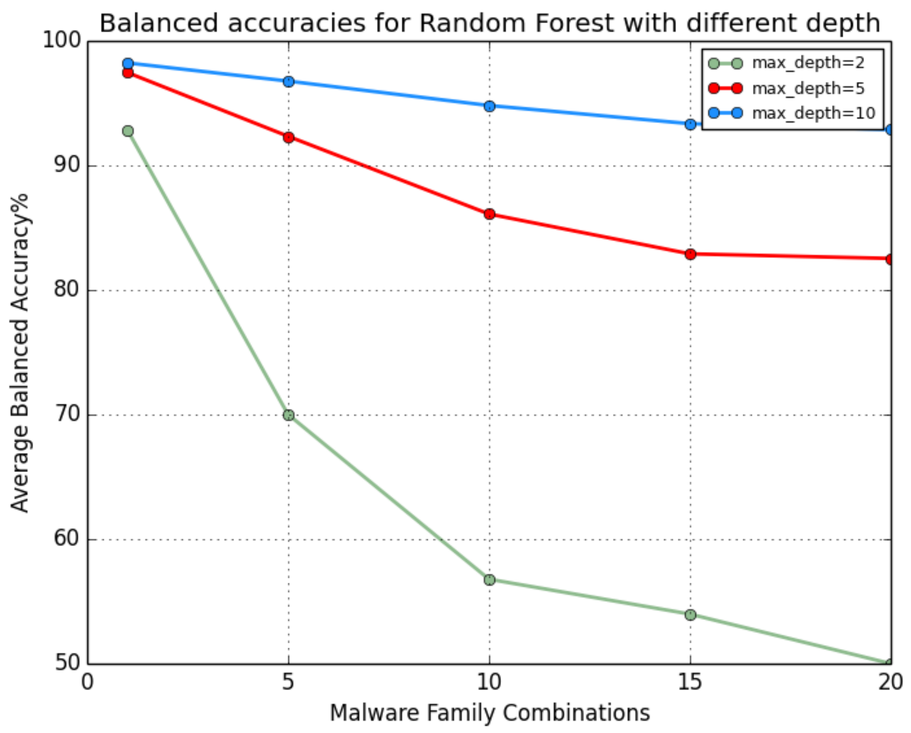


Figure 15: Balanced accuracies for malware combinations at level 1, 5, 10, 15, and 20 using different `max_depth` parameters for random forest

Table 9: Random forest average balanced accuracies for multifamily models

Level	**Accuracy%**	**Level**	**Accuracy%**
1	98.23	11	94.56
2	97.94	12	94.14
3	97.68	13	94.09
4	97.24	14	93.76
5	96.77	15	93.35
6	96.42	16	93.29
7	96.03	17	93.00
8	95.74	18	92.94
9	95.25	19	92.79
10	94.81	20	92.87

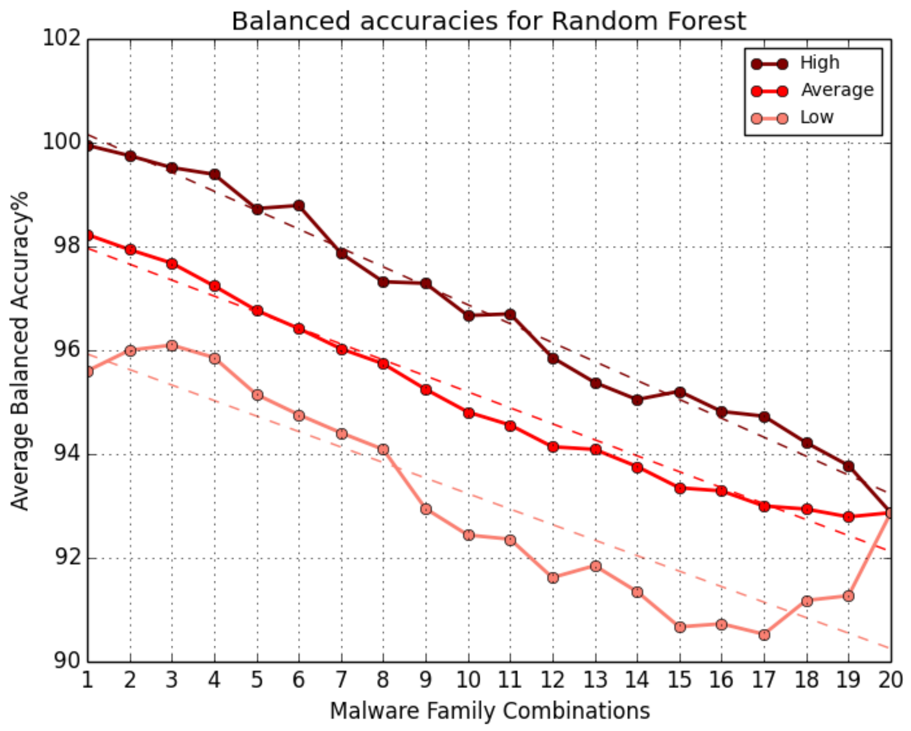


Figure 16: Random forest results for all levels showing average (of 100 experiments), low, and high accuracies and linear fit lines

5.2.4 Multi-layer Perceptron

Table 10 shows how the balanced accuracy for MLP changes as more families are added and as the value of alpha is changed. Figure 17 graphs the data as a line chart. The penalty parameter in MLP is `alpha`. For example, a parameter value of 1e-5 signifies larger weights and a complicated decision boundary [38]. The experiments in Figure 17 indicate that a model with `alpha` value as 1e-5 performed the best. In all three cases, the balanced accuracy dropped significantly in the initial levels. After level 5, the change is subtle.

Table 11 shows a list of accuracies for different combinations of malware families, and Figure 18 graphs the lowest and the highest accuracies at each level. For the high case, the downward trend is more evident. The fact that the high case is also decreasing means that as more families are added, the overall performance strictly decreases. The area between the average line and the low line is greater than the area between the average line and the high line. This means that at each level, there is at least one experiment which has a significantly lower accuracy than its average case.

Table 10: MLP balanced accuracies for changing `alpha`

Level	**20.0**	**10.0**	**1e-5**
1	83.16	91.72	93.97
5	72.91	78.27	84.22
10	70.09	74.71	80.26
15	71.61	76.15	80.07
20	71.96	77.09	80.24

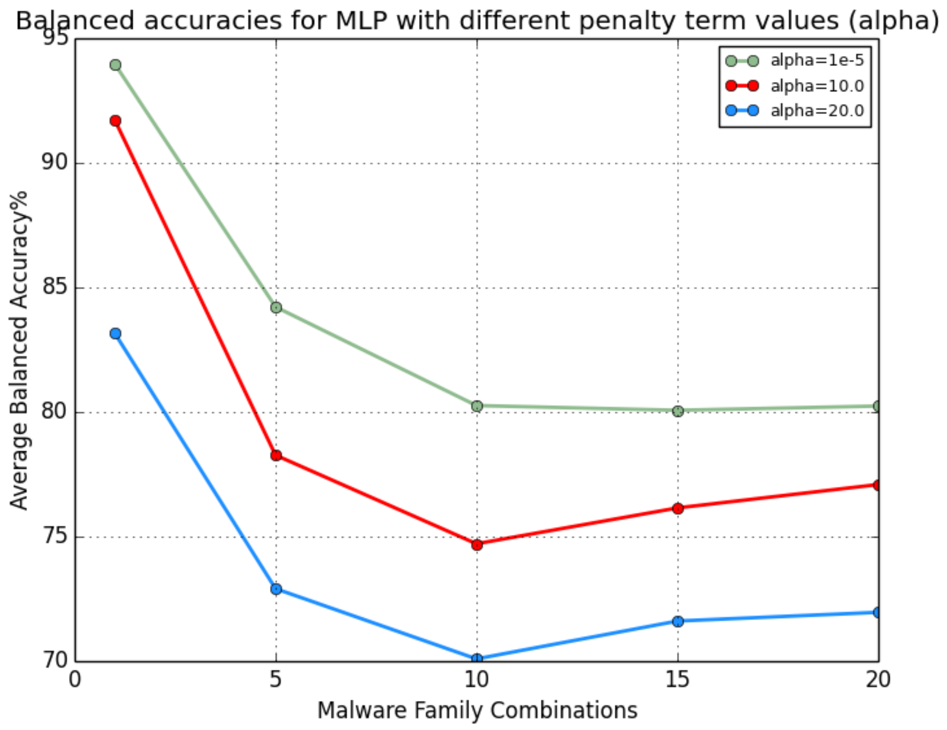


Figure 17: Balanced accuracies for malware combinations at level 1, 5, 10, 15, and 20 using different `alpha` values for MLP

Table 11: MLP average balanced accuracies for multifamily models

Level	Accuracy%	Level	Accuracy%
1	93.97	11	80.59
2	90.45	12	80.27
3	89.10	13	79.92
4	87.29	14	80.36
5	84.22	15	80.07
6	83.63	16	80.37
7	81.82	17	80.14
8	81.47	18	79.82
9	81.06	19	79.45
10	80.26	20	80.24

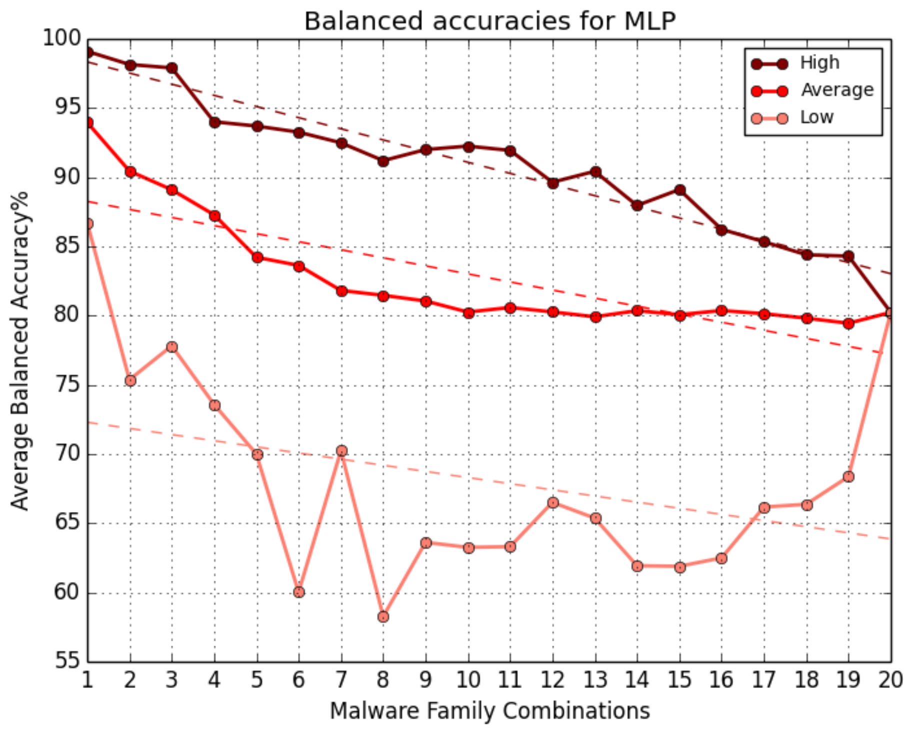


Figure 18: MLP results for all levels showing average (of 100 experiments), low, and high accuracies and linear fit lines

5.3 Overall Results

Figure 19 summarizes the performance of all the techniques mentioned in the previous section. It shows how the average balanced accuracy changes from single-family binary classification experiments to twenty-family binary classification experiments for SVM, k-NN, random forest, and MLP. Generally, the accuracy decreased as more families were added to the malware class. For the single-family case, the average balanced accuracy of random forest was the highest. The same trend followed for every level. The SVM technique resulted in the lowest accuracy at level 1 through level 20. For SVM, there was a drastic drop in the initial levels followed by a constant decrease. For MLP, there was a significant decrease from level 1 to level 7. From level 7, the constant decrease was almost negligible. For k-NN and random forest, level 1 through 10 shows a constant decrease while the trend from level 10 through 20 shows an almost-flat line.

It appears from Figure 19 that SVM and MLP are similar to each other while random forest and k-NN are similar to each other given the parameters specified in Chapter 4. We compare SVM and MLP in Figure 20, and random forest and k-NN in Figure 21. Each technique is split into a separate boxplot with its own scale to provide an enlarged image of the box. These figures represent the results at level 19. Level 19 involves combining 19 malware families into the malware class in order to classify as malware or benign.

For the SVM boxplot, there is one outlier significantly above other points while for the MLP boxplot, there are three outliers lower than the group. The line on the box shows the median. The SVM box is symmetric with most points scattered in the first three quartiles. On the other hand, the MLP points are scattered throughout the boxplot. The mean for the SVM boxplot is 52.42%, and the mean for the MLP boxplot is 79.45%. The maximum value on the y-axis of the SVM boxplot is lesser

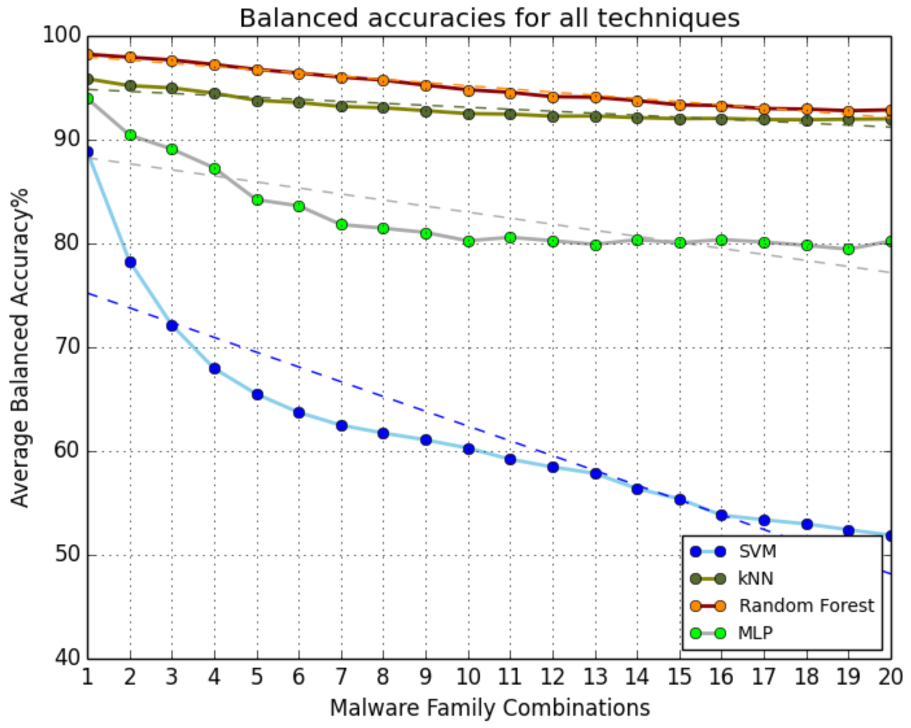


Figure 19: Average balanced accuracies and linear fit

than the minimum value on the y-axis of the MLP boxplot.

The boxplots of k-NN and random forest in Figure 21 show two outliers for k-NN and two outliers for RF that are below their respective groups. The median for the k-NN boxplot is more towards the top of the box while the random forest boxplot has a symmetric box. At level 19, the mean for k-NN is 91.97%, and the mean for random forest is 92.79%.

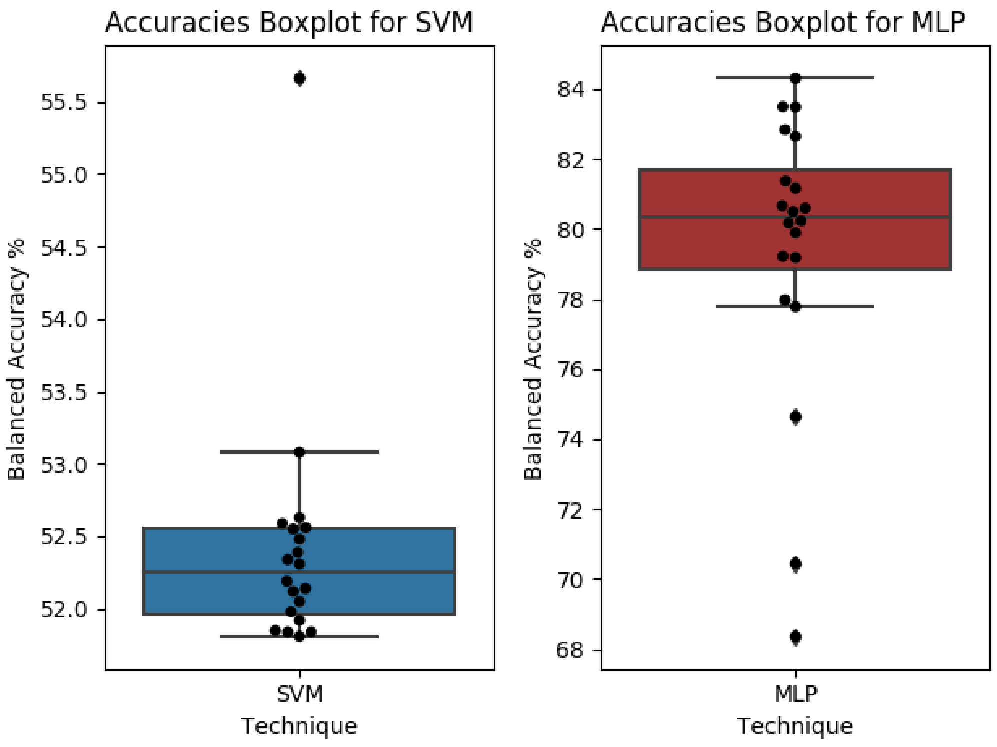


Figure 20: Balanced accuracies for SVM and MLP at level 19

Each point on the graph represents the classification result of malware (consisting of 19 families) and benign.

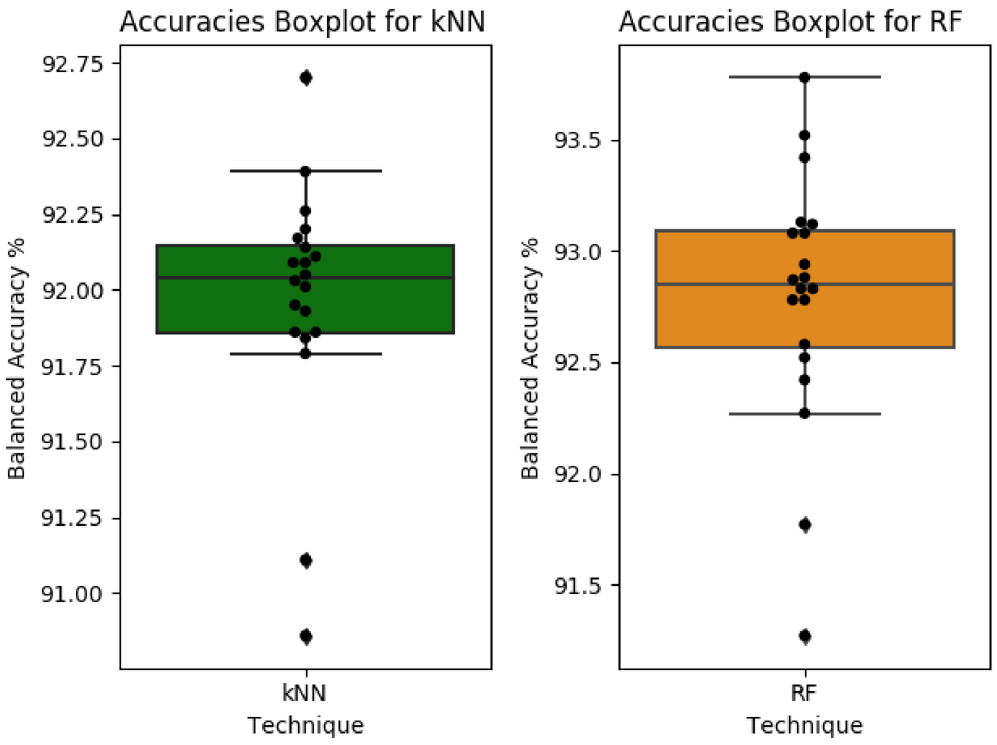


Figure 21: Balanced accuracies for k-NN and RF at level 19

Each point on the graph represents the classification result of malware (consisting of 19 families) and benign.

CHAPTER 6

Conclusion and Future Work

In this paper, we have used bigram features and machine learning techniques to examine how accuracy changes when a model is trained using multiple families. First, we classified each specific family as malware or benign. Then, we trained models using pairs of families, then triples, and all combinations until we reached a model with all 20 families. At the Nth level, we considered $\min(100, \binom{20}{\mathrm{N}})$ experiments.

Figure 22 illustrates a bar graph that compares the performance of a specific model and the most generic model. SVM has the biggest drop from level 1 to level 20. This means that a Linear SVC model was not robust when generic models were involved. MLP performed significantly better than SVM but not better than random forest and k-NN. The parameter of $k = 5$ in k-NN has a very low difference, signifying that k-NN performed very well compared to SVM and MLP. Random forest had the highest accuracy for the most specific case (level 1) as well as the most generic case (level 20) when compared to other techniques. This means that a random forest model with 10 trees and `max_depth` 10 was very robust despite a generic dataset. The low difference between level 1 and level 20 signifies that random forest is a suitable model when generic (multifamily) models are involved. Overall, in every case, adding more families to the malware class lowered the accuracy, and this drop increased as the number of families combined increased. Through each level, the model became more generic and less effective as families were added.

When classifying one specific family, the accuracies are very high indicating the strength of bigrams. In the malware domain, the detection problem is based on classifying a specific family, and in this case, n-grams are a strong feature, as indicated by the accuracies at level 1. When multiple malware families are involved in the binary classification, it would be efficient to have a single model that could

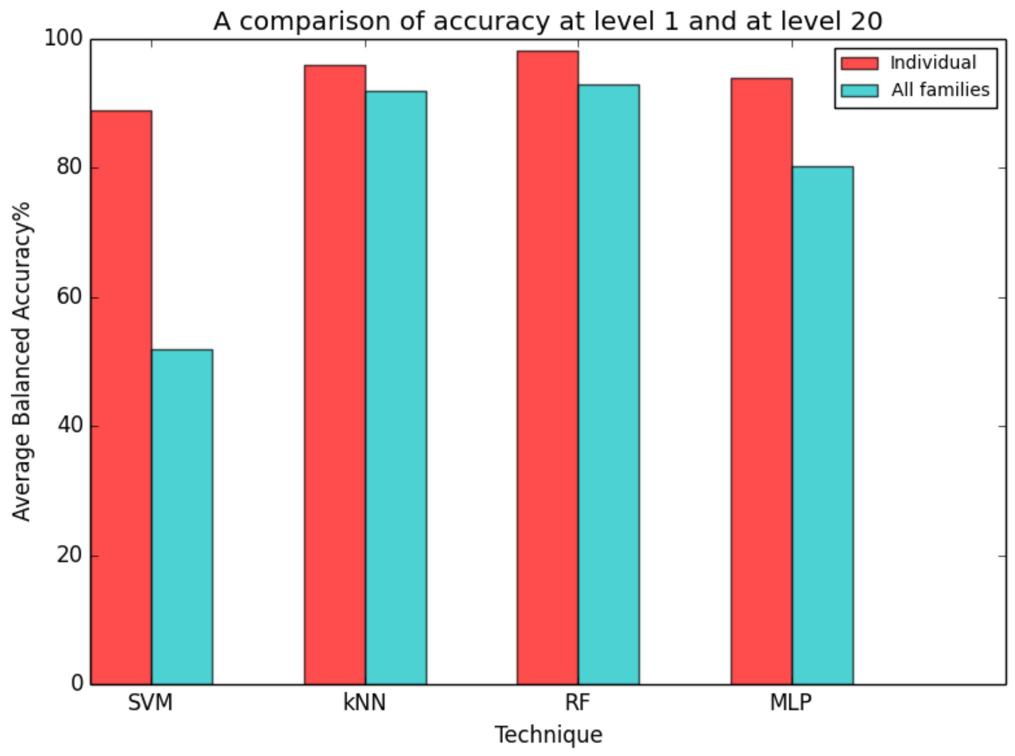


Figure 22: A comparison of average balanced accuracy at level 1 and at level 20 for all techniques

detect multiple families rather than scoring each sample against multiple specific models. Since random forest and k-NN were robust even for a very generic dataset, these techniques could be used to score against a single model trained using multiple families. On the other hand, SVM could not distinguish well between malware and benign files when more than 10 families were involved.

6.1 Future Work

In this research, we have worked with a big dataset and considered 20 malware families with 1000 files in each family. We have used trojans, worms, and other malware types, so our combinations were composed of a variety of malware families. Experiments could be conducted where combinations could include families of the same type, such as adware families. These experiments would show the tradeoff between combining families that are of the same type based on their characteristics versus combining families without any discrimination of types. It would be interesting to see the results of progressively combining 20 adware families, or 20 backdoor trojans, or 20 such specific family types. The challenge in such a research would involve efficiently considering only those families which are similar.

Our experiments included binary classification of malware and benign files. Experiments could also involve multi-class classification. Such experiments that involve classifying and combining multiple malware families help to understand the similarity or dissimilarity between families.

We have classified malware using bigram features. Further research could include experimenting with varying values of n such as 3-grams, 4-grams, and 6-grams. Such experiments would quantify the relationship between higher n and the accuracy of a generic model. To compare and contrast the effectiveness of n-grams, experiments could involve other features such as opcodes. We normalized the training and testing vectors by dividing each frequency by the sum of the frequencies. These vectors could be normalized by the total number of n-grams in the file.

Experiments could be conducted by varying the number of top features. We considered the top 20 features from the malware class, but experiments could consider more than 20 or less than 20 features to examine how feature selection affects accuracy. We have considered four classification techniques: SVM, k-NN, random forest, and MLP. Other techniques could be included such as hidden Markov models and deep neural networks. Clustering techniques such as k-means and model-based clustering would help to compare and contrast clustering and classification techniques.

LIST OF REFERENCES

[1] J. Sahs and L. Khan, ‘‘A machine learning approach to android malware detection,’’ in *2012 European Intelligence and Security Informatics Conference*, ser. EISIC 2012. IEEE, 2012, pp. 141--147.

[2] 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.

[3] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, ‘‘Malware detection based on mining api calls,’’ in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC 2010. ACM, 2010, pp. 1020--1025.

[4] S. Y. Yerima, S. Sezer, and I. Muttik, ‘‘High accuracy android malware detection using ensemble learning,’’ *IET Information Security*, vol. 9, no. 6, pp. 313--320, 2015.

[5] S. Kim, ‘‘PE header analysis for malware detection,’’ *Master's Projects*, 2018, 624.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning.* Springer, 2013, vol. 112.

[7] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning.* New York: Springer, 2001, vol. 1, no. 10.

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

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning.* Cambridge: MIT Press, 2016, vol. 1.

[10] N. Bagga, F. Di 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 2018, INSTICC. SciTePress, 2018, pp. 442--450.

[11] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, ‘‘N-grams-based file signatures for malware detection,’’ in *Proceedings of 11th International Conference on Enterprise Information Systems*, ser. ICEIS 2009, 2009, pp. 317--320.

[12] C. Liangboonprakong and O. Sornil, ‘‘Classification of malware families based on n-grams sequential pattern features,’’ in *2013 IEEE 8th Conference on Industrial Electronics and Applications*, ser. ICIEA 2013. IEEE, 2013, pp. 777--782.

[13] D. K. S. Reddy and A. K. Pujari, ‘‘N-gram analysis for computer virus detection,’’ *Journal in Computer Virology*, vol. 2, no. 3, pp. 231--239, 2006.

[14] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, ‘‘An investigation of byte n-gram features for malware classification,’’ *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1--20, 2018.

[15] H. Zou and T. Hastie, ‘‘Regularization and variable selection via the elastic net,’’ *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301--320, 2005.

[16] A. Nappa, M. Z. Rafique, and J. Caballero, ‘‘The malicia dataset: identification and analysis of drive-by download operations,’’ *International Journal of Information Security*, vol. 14, no. 1, pp. 15--33, 2015.

[17] ‘‘Trojandownloader:win32/adload,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader%3AWin32%2FAdload, accessed: 2019-03-17.

[18] ‘‘Win32/obfuscator,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Obfuscator&threatId=, accessed: 2019-03-18.

[19] ‘‘Trojandownloader:win32/agent,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Agent&ThreatID=14992, accessed: 2019-03-17.

[20] ‘‘Pws:win32/onlinegames,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FOnLineGames, accessed: 2019-03-25.

[21] ‘‘Win32/alureon,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Alureon, accessed: 2019-03-17.

[22] ‘‘Win32/rbot,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Rbot&threatId=, accessed: 2019-03-18.

[23] ‘‘Trojan:win32/bho,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO&threatId=-2147364778, accessed: 2019-03-25.

[24] ‘‘Trojan:win32/startpage,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Startpage&threatId=15435, accessed: 2019-03-18.

[25] ‘‘Backdoor:win32/cycbot.g,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Cycbot.G, accessed: 2019-03-25.

[26] ‘‘Win32/vobfus,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vobfus&threatId=, accessed: 2019-03-18.

[27] ‘‘Pws:win32/delfinject,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/DelfInject&threatId=-2147241365, accessed: 2019-03-17.

[28] ‘‘Win32/vundo,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Vundo&threatId=, accessed: 2019-03-18.

[29] ‘‘Win32/fakerean,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/FakeRean, accessed: 2019-03-17.

[30] ‘‘Win32/winwebsec,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Winwebsec, accessed: 2019-03-18.

[31] ‘‘Adware:win32/hotbar,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Hotbar&threatId=6204, accessed: 2019-03-17.

[32] ‘‘Pws:win32/zbot,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Zbot&threatId=-2147368817, accessed: 2019-03-18.

[33] ‘‘Pws:win32/lolyda.bf,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FLolyda.BF, accessed: 2019-03-25.

[34] ‘‘Trojan.zeroaccess,’’ https://www.symantec.com/security-center/writeup/2011-071314-0410-99, accessed: 2019-03-18.

[35] ‘‘Trojandownloader:win32/renos,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Renos&threatId=16054, accessed: 2019-03-18.

[36] ‘‘Virtool:win32/ceeinject,’’ https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool%3AWin32%2FCeeInject, accessed: 2019-03-17.

[37] A. Liaw, M. Wiener, *et al.*, ‘‘Classification and regression by randomforest,’’ *R news*, vol. 2/3, pp. 18--22, 2002.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, ‘‘Scikit-learn: Machine learning in Python,’’ *Journal of Machine Learning Research*, vol. 12, pp. 2825--2830, 2011.

[39] ‘‘Python software foundation: cPickle: A faster pickle, version 2.7,’’ https://docs.python.org/2/library/pickle.html#module-cPickle, accessed: 2019-03-30.

[40] ‘‘Python software foundation. collections: High-performance container datatypes, version 2.7,’’ https://docs.python.org/2/library/collections.html, accessed: 2019-03-30.

[41] ‘‘Python software foundation. itertools: Functions creating iterators for efficient looping, version 2.7,’’ https://docs.python.org/2/library/itertools.html, accessed: 2019-03-30.

[42] ‘‘Python software foundation. random: Generate pseudo-random numbers, version 2.7,’’ https://docs.python.org/2/library/random.html, accessed: 2019-03-30.

APPENDIX

Additional Results

A.1 Individual Family Results

Red colored bars indicate an accuracy lower than the average while blue bars indicate an accuracy greater than or equal to the average. A darker share indicates farther distance from the average while a lighter shade indicates closer distance to the average.

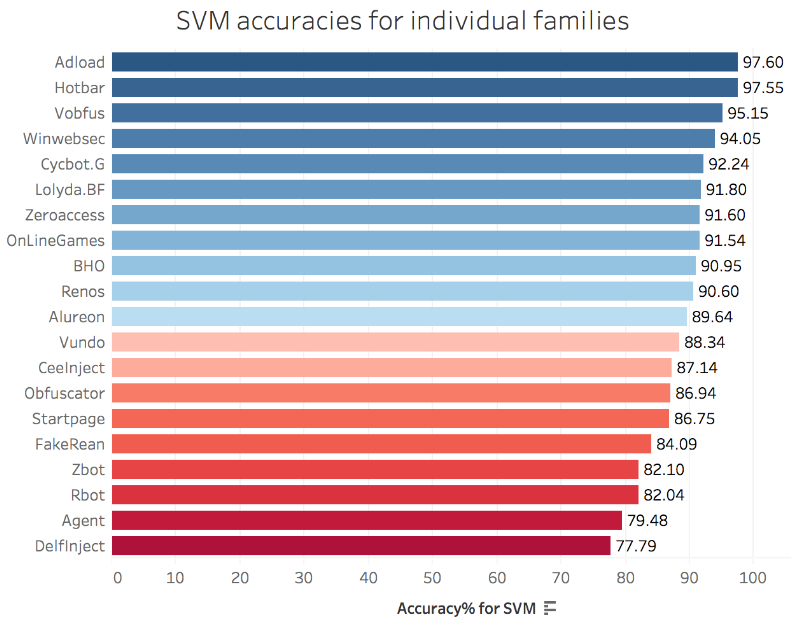


Figure A.23: Individual family accuracy results using SVM

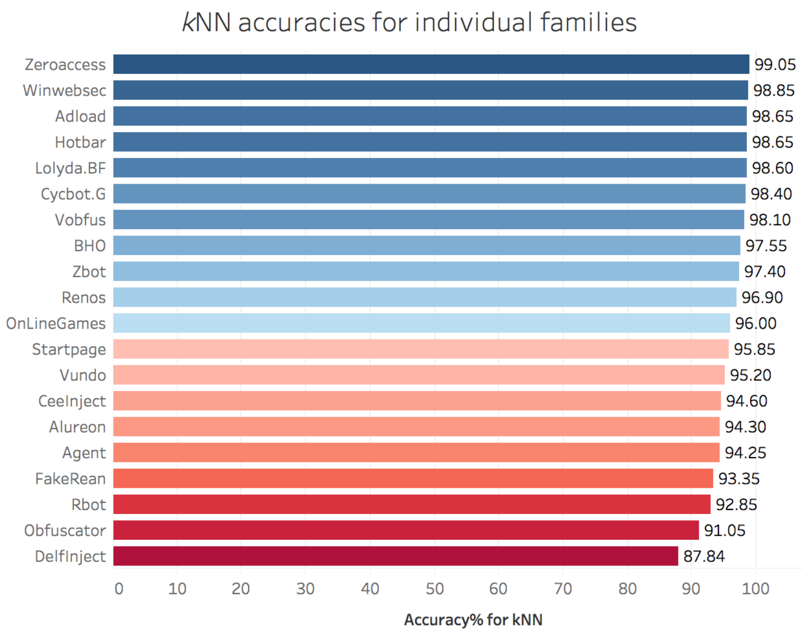


Figure A.24: Individual family accuracy results using *k*-NN

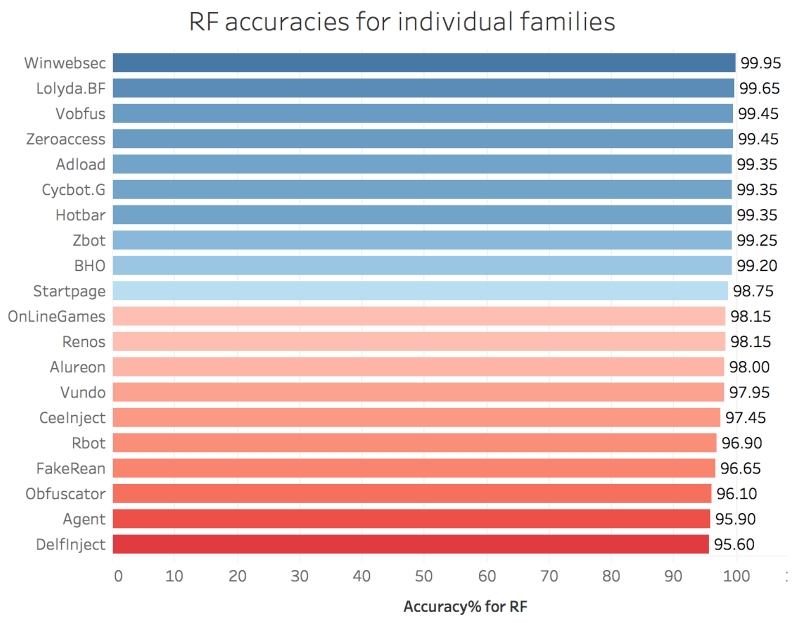


Figure A.25: Individual family accuracy results using Random Forest

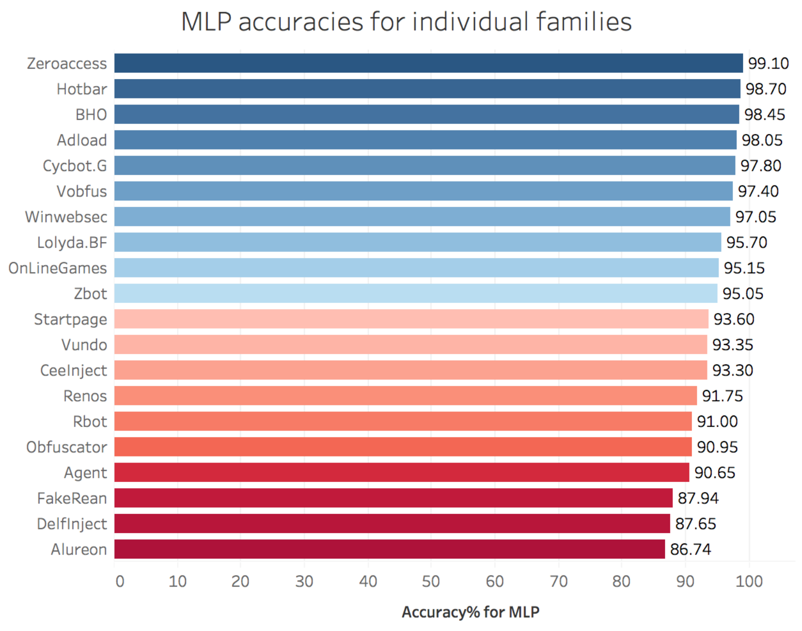


Figure A.26: Individual family accuracy results using MLP