Fall 2017

Measuring the Effectiveness of Generic Malware Models

Naman Bagga
San Jose State University

Measuring the Effectiveness of Generic Malware Models

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Naman Bagga

December 2017

The Designated Project Committee Approves the Project Titled

Measuring the Effectiveness of Generic Malware Models

by

Naman Bagga

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2017

Dr. Mark Stamp	Department of Computer Science
Dr. Jon Pearce	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science

ABSTRACT

Measuring the Effectiveness of Generic Malware Models

by Naman Bagga

Malware detection based on machine learning techniques is often treated as a problem specific to a particular malware family. In such cases, detection involves training and testing models for each malware family. This approach can generally achieve high accuracy, but it requires many classification steps, resulting in a slow, inefficient, and impractical process. In contrast, classifying samples as malware or benign based on a single model would be far more efficient. However, such an approach is extremely challenging—extracting common features from a variety of malware families might result in a model that is too generic to be useful. In this research, we perform controlled experiments to determine the tradeoff between accuracy and the number of malware families modeled.

ACKNOWLEDGMENTS

I would like to thank Dr. Mark Stamp for his constant guidance throughout the course of the project. I would also like thank my committee members Dr. Jon Pearce and Dr. Thomas Austin for reviewing my work and providing valuable feedback.

I am also grateful to my family and friends who have supported me throughout the course of my Master's program. Finally, I wish to thank my coworkers without whom this journey wouldn't be possible.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is short for "malicious software" and is defined as software that is intended to damage computers and computer systems without the knowledge of the owner [1]. According to Symantec [2], more than 357 million new variants of malware were found in 2016. Detecting all these new viruses is clearly an important research topic

The complexity of malware is growing rapidly making malware detection a critical issue. Various malware detection techniques like signature-based detection and heuristic detection have been developed over time to combat this issue. Signature-based detection is the primary method used by commercial anti-virus products to detect malware. This method involves storing signatures for all the different known malware samples and comparing them to the file being scanned.

Signature-based detection is effective in general but fails to detect metamorphic malware as such malware can modify their signature. Machine learning techniques like hidden Markov models (HMM) [3], support vector machines (SVM), k-Nearest Neighbors (k-NN), decision trees and random forests [4] are used for effectively detecting metamorphic malware. These models are usually trained using static file features like opcode sequence and byte n-grams.

Previous research [5, 6, 7, 8] has shown that n-grams can be used for malware detection with very high accuracy. However, a recent research study [9] on n-grams rejects this claim and argues that n-grams promote over-fitting and the resulting models only detect whether a file is a windows executable. This research used a large

dataset consisting of all kinds of malware to build a single classifier. This approach is questionable since malware detection is usually considered a problem specific to a particular kind of malware and not just any general malware. The claim about the ineffectiveness of n-grams needs to be investigated further by experimentation. The feasibility of the idea of building a generic malware classifier to detect any malware also needs to be studied.

In this research, we perform experiments using different machine learning techniques like SVM, k-NN, random forests and statistical techniques like chi-squared test. We also explore the effectiveness of two different features: opcodes and n-grams. We aim to determine the tradeoff between accuracy and the number of malware types modeled by conducting a set of experiments on individual malware families and the same set of experiments on a combination of various malware families.

The remainder of the paper is organized as follows. In Chapter 2, we discuss relevant background information, related work and the motivation for conducting this research. Chapter 3 discusses the machine learning techniques used — SVM, χ^2 test, k-NN and random forests. We discuss these techniques briefly along with experimental setup and datasets used. The experiments and results are described in Chapter 4. The paper concludes in Chapter 5 where we also discuss possible future work related based on this research.

CHAPTER 2

Background

2.1 Malware Classification

Malware can be broadly classified into three types based on concealment strategy: no concealment, encrypted, and metamorphic [10].

- **No concealment malwares** are the ones that do not use any techniques to hide their code or alter the statistical properties of the files. Such malware can easily be detected by simple techniques like signature-based detection.
- **Encrypted malwares** encrypt certain parts of the file that contain the malicious code. These can be detected by executing them in a sandbox, since on execution, the malware decrypts the encrypted part.
- **Metamorphic malwares** are the ones that carry a morphing engine within their body that can modify their structure. These are the ones that are the most difficult to detect as they can easily evade signature based detection.

Metamorphic malwares have been around since the late 1990's. These were developed to defeat common malware detection techniques like signature-based detection. Previous work has shown that machine learning techniques like HMM are an efficient way to detect such malware. These techniques have been tested using both static features like n-grams and dynamic features like API calls [11]. This is because metamorphic malware engines rely on code transposition and obfuscation to modify their signature; however, the behavior of the virus and the sequence of API calls it uses is the same.

2.2 Related Work

In this section, we talk about some of the previous work that has been done in this area based on which we chose the techniques and features for our experiments.

Wong and Stamp [12] show that opcodes are effective features when used with hidden Markov models. They trained models for three common metamorphic malware families and were able to successfully detect them. This malware detection scheme outperformed most of the commercial anti-virus products. The results are not surprising considering the fact that most commercial products rely on signature based detection methods. This makes a strong case for opcodes since they are directly related to the functionality of the program.

Singh et al. [13] applied three different techniques to obtain three scores: HMM score, simple substitution distance score and opcode graph score. These three scores were combined using a SVM classifier to build a stronger and more robust classifier. This research demonstrates the use of SVM for combining scores from different classifiers. Support vector machines work well when the data has a large number of features as the technique projects data into a higher dimension. The approach used here is scalable since it would work even if additional scores or features are added. These ideas can be further generalized to using SVMs on n-gram frequencies to classify malware. This can be done by selecting k n-grams and using their frequencies as features for each file.

Annachhatre et al. [14] trained several HMM's on compilers and malware generators and combined these with clustering to devise an effective malware classification technique. This research demonstrated the use of HMM and clustering to effectively classify previously unknown malware.

Reddy and Pujari [5] use a feature selection method that ranks n-grams based on frequency and entropy. Their experiments show that performing a class-wise feature selection improves the efficiency of the models. This process involves extracting the top k n-grams from the benign and malware set and using a union of the two sets as the feature space. This process can be used as a benchmark for future work.

Raff et al. [9] performed experiments on n-grams using elastic-net regularized logistic regression on a dataset containing over $200,000$ malware samples. The results obtained from these experiments were poor as the classifier had very low accuracy. This data was obtained from an undisclosed industry partner making this study hard to validate. Similar studies using publicly available datasets should be of great advantage.

One of the ways to compare the performance of generic malware detection to specific malware detection is to perform a set of experiments on individual malware families and then on the combinations of those families. Accuracy, precision, and recall can be used as metrics to quantify the effectiveness of these models. This should give a direct comparison of the generic and specific malware detection approaches.

CHAPTER 3

Implementation

In this chapter we first discuss the dataset used in this project and then cover the methods used for conducting the experiments.

3.1 Dataset

The following eight malware families were part of the Malware dataset for this project.

Gatak [15] is a Trojan that collects information about the infected PC and sends it to a hacker. This Trojan hides itself as part of a key generator application or an update for a legitimate application.

Kelihos [16] is a family of Trojans that send out spam emails with links to installers of Kelihos malware. It is a botnet that communicates with remote servers to send spam emails and capture sensitive information.

Lollipop [17] is an adware program that shows the user ads while browsing the web. This adware can also redirect search engine results, monitor the user's activity and send information to a hacker.

Obfuscator.ACY [18] is a family of obfuscated viruses that hide their purpose through obfuscation. The underlying malware could have just about any purpose.

Ramnit [19] is a worm that spreads through infected removable drives. It is capable of stealing sensitive information like bank credentials and can give a hacker access and control of your PC.

Winwebsec [20] is a Trojan that pretends to be an antivirus product. It convinces

the user to pay for the fake product by displaying messages stating that the computer has been infected.

Zbot [21] or Zeus is a Trojan horse that infects systems by downloading configuration files or updates. It is a financial botnet that steals confidential information like online credentials

Zeroaccess [22] is a Trojan horse that makes use of a rootkit to hide itself. ZeroAccess creates a backdoor on compromised systems and is capable of downloading additional malware

These malware samples were obtained from the Microsoft Malware Classification Challenge [23] and the Malicia Project. Table 1 lists the number of samples used from each malware family. Table A.10 lists all the 300 benign executables used from System32 in the benign set. These were collected from a 32 bit system running a fresh install of Windows XP.

Table 1: Malware Samples

Malware Family	Number of Samples
Gatak	1013
Kelihos	2942
Lollipop	2478
Obfuscator.ACY	1228
Ramnit	1541
Winwebsec	5820
Zbot	2167
Zeroaccess	1306

In the following sections we will describe the techniques used to perform experiments on these datasets.

3.2 Support Vector Machines

The first set of experiments conducted used support vector machine (SVM) as the machine learning technique and n-grams as the feature.

3.2.1 Support Vector Machines

SVM is a supervised learning algorithm that is used for binary classification. SVM takes a set of labeled data as input and tries to find the ideal separating hyperplane between the two classes. Once the model is trained, any new data can be classified based on the model.

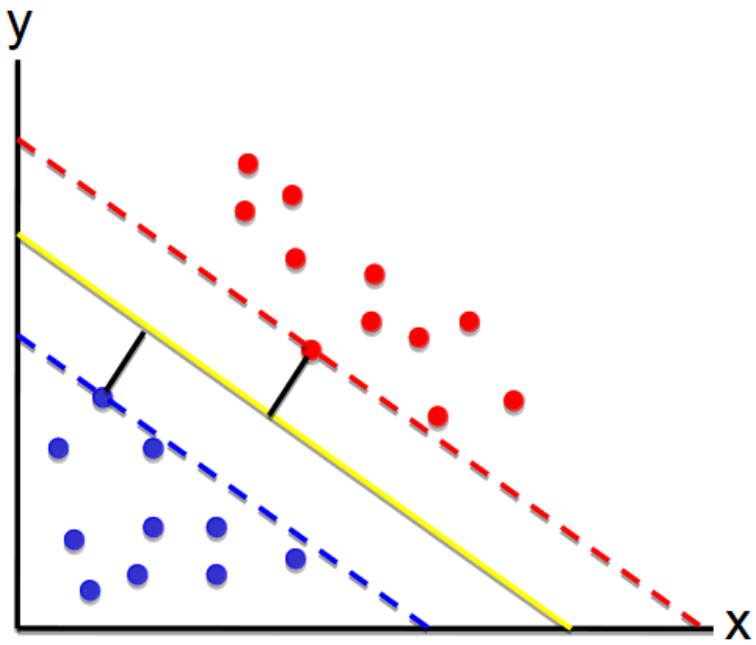


Figure 1: SVM - Separating Hyperplane

There are four big ideas [24] behind SVMs-

- Separating Hyperplane — The training data is separated into two classes using a hyperplane. If the input data is N dimensional, the hyperplane has a

dimensionality of $N - 1$.

- Maximize the margin — The separating hyperplane is chosen such that the margin between the two classes of data is maximized. This gives us the best odds of classifying the new data correctly and prevents any bias in the classifier. Figure 1 depicts the separating hyperplane for the given data.
- Work in a higher dimensional space — In SVMs, the data is projected to a higher dimensional space to make it easier to find the separating hyperplane. This is because working in a higher dimensional space gives extra dimension to the hyperplane as well. This is unlike most machine learning techniques where dimensionality reduction is favorable.
- Kernel trick — To transform data to a higher dimension, a *kernel function* is used. Picking the correct kernel function is tricky. The most common kernel functions used are - Polynomial learning machine, Gaussian Radial-basis funciton (RBF), and Two-layer perceptron.

3.2.2 Implementation

The SVM based experiments used n-grams as features. Each file in the malware dataset was read as a binary file and every sequence of n bytes was used as a n-gram. The frequencies of these n-grams are used as features. The n-gram frequencies for each file (malware and benign samples) form our feature vectors. The SVM based experiments were conducted with the help of the scikit-learn [25, 26] Python library. SVM models are trained on a subset of the dataset and tested on the remainder. More details about the experiments conducted will be discussed in Section 4.2.1.

3.3 Chi-squared test

Chi-squared test [27] of goodness is a popular statistical technique that can be applied to categorical data to check if the observed frequency distribution differs from the theoretical frequency distribution. This is one of the oldest statistical techniques discovered by Pearson [28] in 1900.

Mathematically, the χ^2 statistic is essentially a normalized sum of square deviation between the expected and the actual frequency distributions. It is computed as

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

where

χ^2 = Cumulative chi-squared statistic

n = number of dimensions or features

O_i = observed value of the i^{th} feature

E_i = expected value of the i^{th} feature

The χ^2 statistic was implemented in python to perform a set of experiments on the n-gram distribution in the files. More details about the experiments will be discussed in Section 4.3.1

3.4 k-Nearest Neighbors

The k-Nearest Neighbor (k-NN) algorithm is the simplest machine learning algorithm possible [24]. k-NN is a supervised learning algorithm and requires labelled training data. k-NN classifies a point x based on the k points in the training set that are nearest to it. The advantage of using k-NN is that it involves no training as

labeled training data is all that the algorithm needs.

Several variations of k-NN are possible. One possibility is to weigh the nearest neighbors based on their distance from the point. This is a good idea since one point very near to the point can get more weight than two points a little far from it. Another possible approach is to weigh the nearest neighbors based on their relative frequency in the training set. This is useful when there is an imbalance in the match and no-match set.

In this research, k-NN was implemented using python to build classifiers based on n-gram frequencies as features. More details about these experiments and their results will be discussed in Section 4.4.1

3.5 Random Forests

Random forests are a generalization of decision trees [24]. Random forests use bagging on the observations as well as the features to construct multiple decision trees. A random subset of the observations and a random subset of the features is picked to construct multiple decision trees. For classification, the outputs of all these decision trees are combined using some technique like a majority vote to get a single classification.

The key advantage of random forests is that each decision tree uses a random subset of observations and features making them highly immune to overfitting. The obvious drawback is that we lose the simplicity of a single decision tree.

Random forests and k-NN classifiers are somewhat similar in the sense that both of them rely on neighborhood based classification. However, in the context of random forests, neighborhood refers to the collection of decision trees that it comprises of.

In this research, we used scikit learn's ensemble library [25] for python to implement random forests. n-gram frequences were used as features for these experiment which will be described in more detail in Section 4.5.1.

CHAPTER 4

Experiments and Results

This chapter discusses the experiments performed using different techniques and explains the results. The techniques used are — SVM, χ^2 test, k-NN, and random forests. We discuss the experimental design and metrics used to measure success as well.

4.1 Experimental Design

Each of the experiments performed used n-gram frequencies as the features. Malware samples listed in Table 1 were used as the malware set and benign dataset listed in Table A.10 was used as the benign set. Two set of experiments were conducted using each technique. The first set of experiments was conducted using all the available malware samples of each family. The second set of experiments used a more balanced dataset as only 1000 samples from each of the eight malware families were used for these experiments.

For the purpose of these experiments we select $n = 2$ i.e. we use 2-byte n-grams or bigrams as features. The total number of possible bigrams is 2^{32} or 65536. Using these many features would be very time consuming and ineffective. Hence, a feature selection mechanism needs to be implemented to perform these experiments. The feature selection approach used here is similar to the one used by Reddy and Pujari in [5]. The 10 most frequent bigrams present in the malware set are chosen along with the 10 most frequent bigrams present in the benign set. A union of these two sets is used as the feature set for these experiments. Figure 2 describes the feature selection process flow.

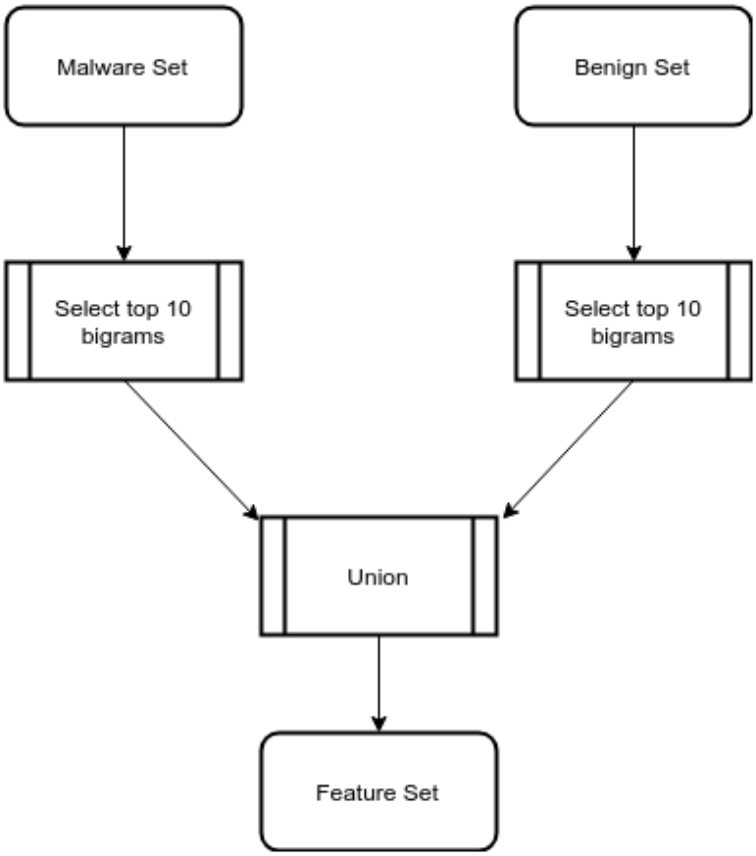


Figure 2: Bigram Feature Selection Process

To validate the results obtained from bigrams, experiments were also conducted using $n = 3$ and $n = 4$ i.e. using 3-grams and 4-grams as features. Five-fold cross validation was used for each of the experiments performed. This implies the dataset was then divided into five parts each of which served as the test set once. When these malware families were combined, special care was taken to ensure that each of these five folds had a similar distribution of samples from each family.

Most of the machine learning techniques used in this research are classification techniques as they return a classification of the tested files as the output. To measure

the effectiveness of the models, balanced accuracy was computed using the equation

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{TN}+\text{FP}}\right)$$

Here, the symbols TP, TN, FP, FN are defined as:

- True Positive (TP) — Number of malware samples that are correctly classified as malware
- True Negative (TN) — Number of benign samples that are correctly classified as benign
- False Positive (FP) — Number of benign samples that are misclassified as malware
- False Negative (FN) — Number of malware samples that are misclassified as benign.

The rationale behind using balanced accuracy is to deal with the imbalance in the number of samples in the malware and benign sets. The imbalance is unavoidable since the size of the malware dataset varies depending on the number of malware families combined. Measuring the balanced accuracy ensures that the accuracy is penalized if the number of false positives is large. A low false positive rate is extremely important for malware detection since users lose confidence if a benign file that the user is sure about is misclassified as malware.

4.2 SVM

In this section we discuss the experiments and results for the SVM and n-gram based approach.

4.2.1 Experiments

For the SVM based experiments, we first select the n-grams to use using the technique mentioned in Section 4.1. Once we have the features i.e. the n-grams to be used, the frequency for each of the n-grams is computed for each file. These frequencies are normalized as a percentage of the total n-grams in each file to make this number independent of file size. The matrix of these scaled frequencies forms the feature vectors.

Malware samples listed in Table 1 were used as the malware set and benign dataset listed in Table A.10 was used as the benign set. An SVM was trained on these datasets using a radial basis function (RBF) kernel and soft margin (C) = 1.

These experiments were first conducted for all eight malware families. After this, these families were combined in all possible combinations of 2, 3, 4 ... 8 and these experiments were repeated on the resultant datasets.

4.2.2 Results

The accuracies for experiments conducted on individual malware families are listed in Table 2. Six of the eight malware families were classified with a very high accuracy. The other two families — Lollipop and Ramnit are classified with a lower accuracy. We also observed that results for both sets of experiments — using all samples and 1000 samples are similar. This is as expected because 1000 samples can be considered large enough to be representative of the entire set of a particular family. Since most of the malware families score pretty well individually, it would be interesting to look at all possible combinations of them to see how the generic malware models behave.

Table 2: SVM - Accuracies for individual families

Malware Family	Balanced Accuracy	
	All Samples	1000 samples
Gatak	0.9630	0.9803
Kelihos	0.9983	0.9958
Lollipop	0.8409	0.8757
Obfuscator	0.9199	0.9195
Ramnit	0.8402	0.8619
Winwebsec	0.9712	0.9705
Zbot	0.9614	0.9627
Zeroaccess	0.9793	0.9808

Table 3 lists the average results of the experiments conducted on all possible combinations of the eight malware families. Here "Combine k" refers to the average accuracy of all experiments where k malware families were combined. Refer to Table 9 for a complete list of accuracies.

Table 3: SVM - Average accuracy for malware models

Malware Families	Average Balanced Accuracy	
	All Samples	1000 samples
Individual	0.9342	0.9434
Combine 2	0.8806	0.9010
Combine 3	0.8402	0.8605
Combine 4	0.8081	0.8310
Combine 5	0.7827	0.8077
Combine 6	0.7603	0.7890
Combine 7	0.7384	0.7738
Combine 8	0.7265	0.7678

The average accuracy is plotted against the generalization of the malware dataset in Figure 3. Looking at the plot, we see a significant drop in accuracy with the increase in generalization of datasets. When models are trained on individual malware families,

the balanced accuracy is about 93% which drops to about 73% for the generic model trained on all the eight families. This is intuitive since as the models become more and more generic, they tend to learn very little about the data. As a result, we get models that have very little margin between the two classes.

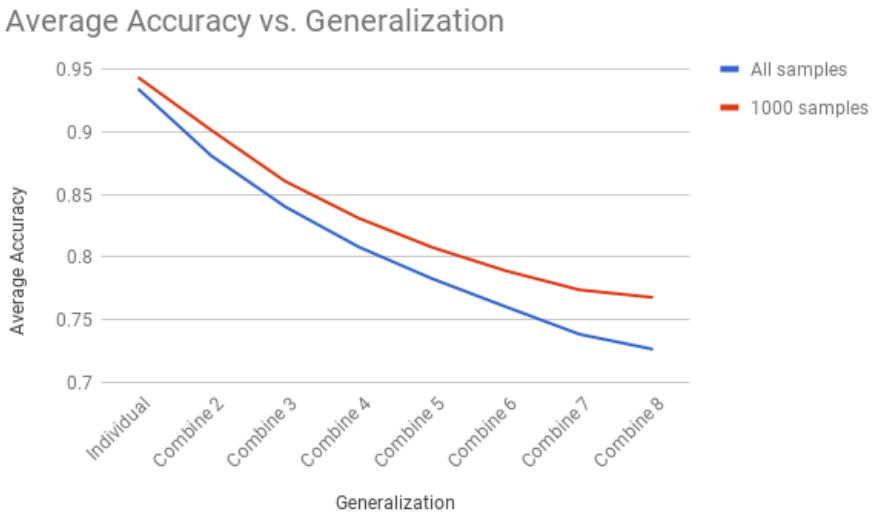


Figure 3: SVM Results - Average accuracy for generic models

To validate these results, the bigrams were examined to see if they followed any pattern. Table 4 lists all the n-grams used by the most generic model which was trained on the combination of all malware families. About half of the bigrams consist of non-printable characters. Among the remaining bigrams, a fraction of them consist of alphabets indicating that the bigrams here do not have any commonly occuring strings in them.

`0x0000` is a common bigram as it represents null which is common in large executable files. Another common noticeable bigram in the list is `0xFFFF` which has all bits set. A lot of the other common bigrams that were used have one byte as `0x00` showing that null bytes are pretty common in the files.

Table 4: Most frequent bi-grams

0000	FFFF	0100	0101
0303	4200	00FF	4000
008B	0200	4100	0001
2000	0020	CCCC	6500
0065	0074	7400	FF75

These experiments were also repeated for 3-grams and 4-grams with similar results. Refer to Appendix B for results of those experiments.

4.3 Chi-squared test

In this section we discuss the experiments and results for the χ^2 statistic based approach.

4.3.1 Experiments

For the experiments using χ^2 statistics for n-grams, the malware samples listed in Table 1 were used along with the benign samples listed in Table A.10. Like the previous n-gram experiments, we initially used $n = 2$ for these experiments. The experiments were repeated for $n = 3$ and $n = 4$ as well. The feature selection method was the same as depicted in Figure 2.

Once the features have been selected, for each malware sample in the training set, we count the number of bigrams of each type and add them. These cumulative frequencies are then normalized as a percentage of the total n-grams to get the expected distribution of n-grams in our training data. This expected distribution forms our model to score against.

In the testing phase, χ^2 statistic was computed for each malware and benign

sample in the test set. These χ^2 values are used as scores for the respective samples. ROC curves were plotted using these scores to evaluate the effectiveness of these classifiers. These experiments were performed on all combinations of 1,2,3 ... 8 malware families.

4.3.2 Results

First we look at how well the scores performed when trying to classify individual malware families. Table 5 lists the AUC values for each of the eight malware families.

Table 5: χ^2 test - AUC ROC for individual families

Malware Family	AUC ROC	
	All Samples	1000 samples
Gatak	0.8784	0.9921
Kelihos	0.9943	0.9930
Lollipop	0.6541	0.6876
Obfuscator	0.8750	0.8712
Ramnit	0.8772	0.8748
Winwebsec	0.9450	0.9384
Zbot	0.8709	0.8646
Zeroaccess	0.9502	0.9472

Most of the families could be detected successfully using the χ^2 scores with the exception of Lollipop which gave an AUC of 0.65. Figure 4 depicts the ROC curve for Lollipop. This observation is similar to the SVM results where Lollipop had a lower balanced accuracy than rest of the families.

Most of the other malware families could be detected easily. Specifically, Kelihos was the easiest to detect (refer to ROC curve in Figure 5). The AUC for Kelihos is 0.99 which implies that χ^2 scores could distinguish the family from benign files very accurately.

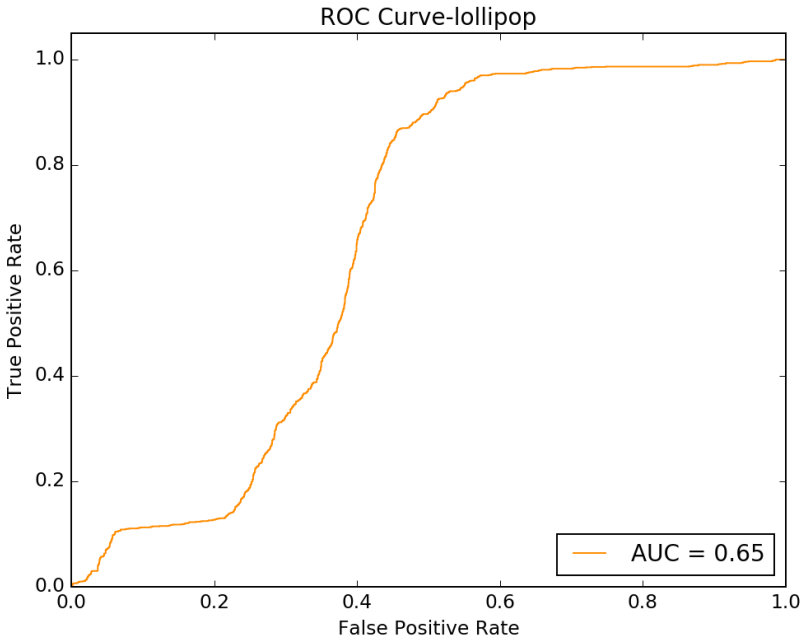


Figure 4: χ^2 ROC Curve - Lollipop

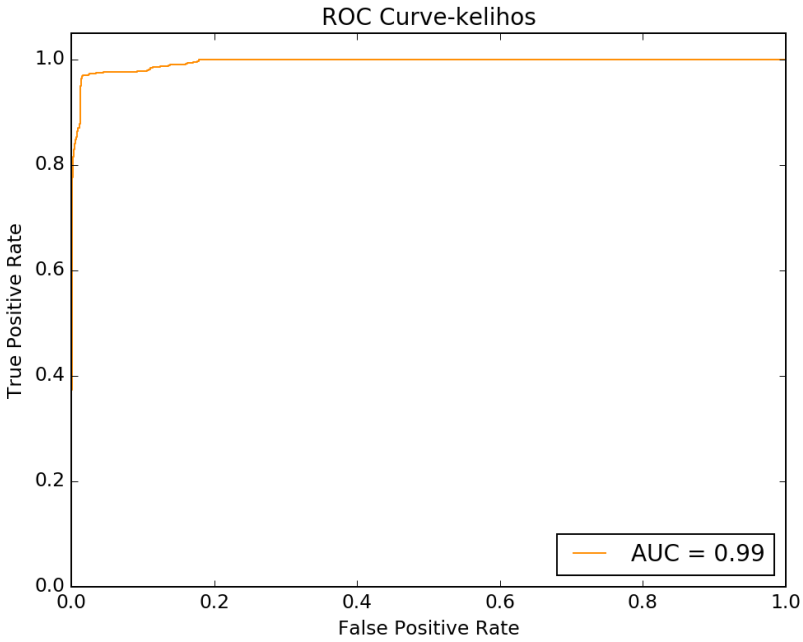


Figure 5: χ^2 ROC Curve - Kelihos

Now we analyze the results of the experiments that modeled more than one family at a time. The average AUC is plotted against the number of families combined in Figure 6. We can see a constant drop in AUC with increase in the number of families combined however the magnitude of this drop is smaller than what was observed for SVM. Even the most generic malware model yields an AUC of 0.80 which is pretty decent considering the model was trained on all eight families. Figure 6 also shows a little deviation between the results for the two sets of experiments. The experiments conducted using only 1000 samples of each malware family show a sharper drop in AUC which becomes constant after that. This is probably because the 1000 samples chosen randomly from each family might be slightly different statistically.

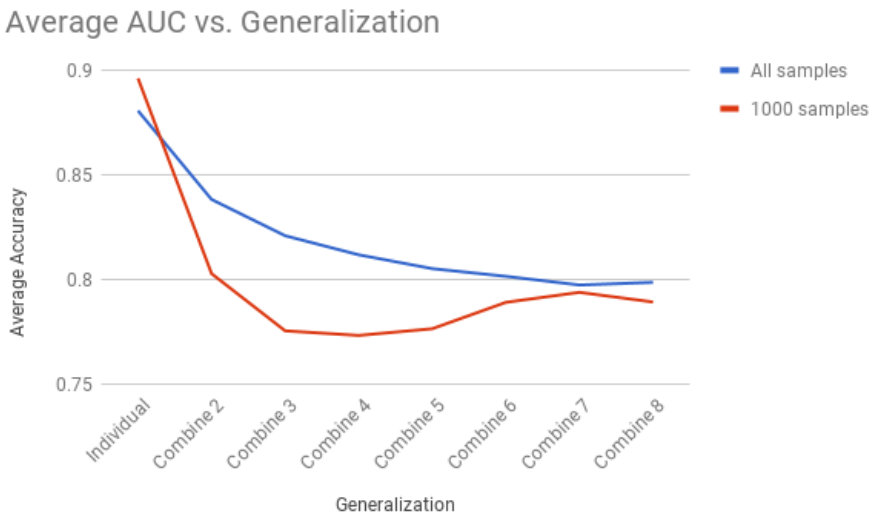


Figure 6: χ^2 Results - Average AUC for generic models

The results for the experiments conducted using 3-grams and 4-grams were similar to these results. These results can be found in Appendix B.

4.4 k Nearest Neighbors

In this section we discuss the experiments and results for the k-NN based approach.

4.4.1 Experiments

For the experiments using k-NN, malware samples listed in Table 1 are used along with benign samples from Table A.10. Like previous experiments, we used $n = 2$ initially and the feature selection method used was the same as depicted in Figure 2. These experiments were repeated for $n = 3$ and $n = 4$ as well.

After the n-grams are selected, we compute their frequencies and normalize them as a percentage of the total n-grams int the sample. Since k-NN does not require training, these labeled points from the training set form the model.

The value of k was picked by performing some experiments on the family Gatak. k was varied from 2 to 10 and the balanced accuracy was computed for all these cases. Figure 7 depicts the results of these experiments.

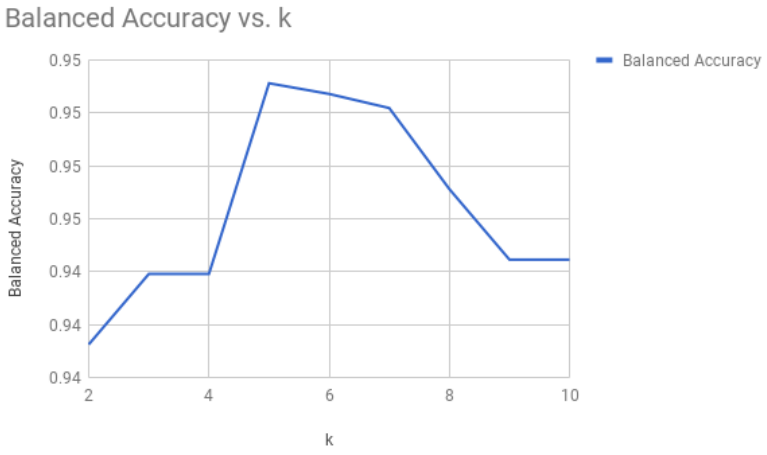


Figure 7: k-NN Results - Varying values of k

Based on experimental results, we pick $k = 5$ since it gives us the highest balanced accuracy. Also, the value 5 is neither too small nor too large to cause misclassification. The neighbors are weighted based on distance which is an intuitive thing to do

n-gram frequencies were computed for each malware and benign sample in the test set and the resultant vectors are fed into the k-NN classifier. Five-fold cross validation was used for these experiments.

4.4.2 Results

First we look at the accuracies for individual families. Table 6 summarizes the results for those experiments. Using k-NN all families score a very high balanced accuracy except Ramnit which is classified with an accuracy of about 86%. The results for the two sets of experiments are similar.

Table 6: k-NN - Accuracy for individual families

Malware Family	Balanced Accuracy	
	All Samples	1000 samples
Gatak	0.9514	1.0000
Kelihos	0.9765	0.9852
Lollipop	0.9334	0.9438
Obfuscator	0.9211	0.9267
Ramnit	0.8632	0.8752
Winwebsec	0.9481	0.9487
Zbot	0.9525	0.9600
Zeroaccess	0.9799	0.9827

Figure 8 depicts the variation in average accuracy with increasing generalization of the models. The average accuracy drops consistently when the number of families combined are increased. The magnitude of this drop in accuracy is smaller than what is observed for SVM and χ^2 test. The most generic model that combines all

eight families still has a balanced accuracy of 86%. The drop in accuracy however is considerably large for all practical purposes.

The results for experiments conducted using 3-grams and 4-grams as features were similar. Refer to Appendix B for the results of those experiments.

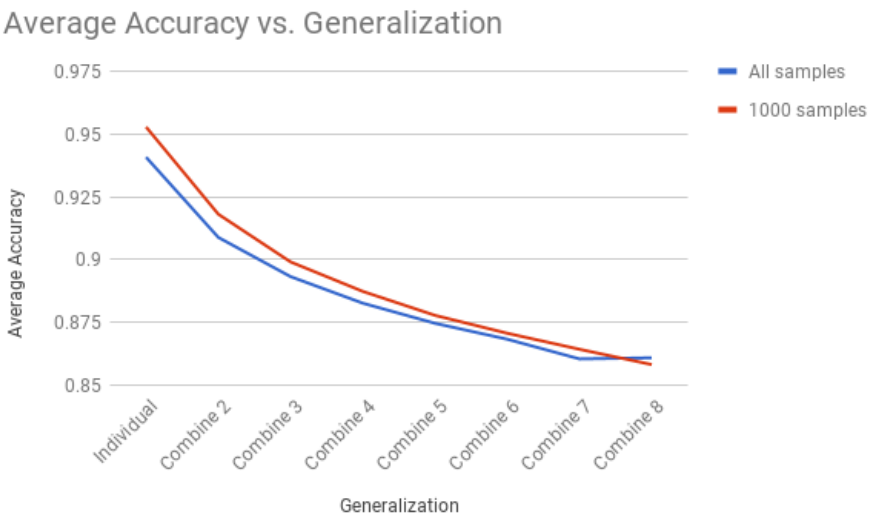


Figure 8: k-NN Results - Average accuracy for generic models

4.5 Random Forests

In this section we discuss the experiments and results for the random forest based approach.

4.5.1 Experiments

For the experiments using random forests, malware samples listed in Table 1 are used along with benign samples from Table A.10. Like previous experiments, we conducted experiments for $n = 2$, $n = 3$ and $n = 4$. The feature selection method used was the same as depicted in Figure 2.

After the n-grams are selected, we compute their frequencies and normalize them as a percentage of the total n-grams in the sample. These are then used to train a random forest classifier. For these experiments we pick the number of decision trees in the forest to be 10. The criterion to measure the quality of the split of the decision trees is entropy or information gain.

n-gram frequencies were computed for each malware and benign sample in the test set and the resultant vectors are fed into the random forest classifier. Five-fold cross validation was used for these experiments.

4.5.2 Results

First we look at the accuracies for individual families. Table 7 summarizes the results for those experiments. Using random forests all families score higher than a 90% balanced accuracy.

Table 7: Random Forests - Accuracy for individual families

Malware Family	Balanced Accuracy	
	All Samples	1000 samples
Gatak	0.9882	1.0000
Kelihos	0.9982	0.9962
Lollipop	0.9736	0.9750
Obfuscator	0.9505	0.9407
Ramnit	0.9049	0.9079
Winwebsec	0.9897	0.9882
Zbot	0.9894	0.9765
Zeroaccess	0.9887	0.9922

Figure 9 depicts the variation in average accuracy with increasing generalization of the models. The average accuracy drops gradually with increasing number of families being modeled. The magnitude of the drop is even smaller than what was

observed for k-NN as the most generic model has an accuracy of 88%. The bagging techniques used by random forests are probably the explanation for why the classifiers are significantly stronger in this case. For instance if we think about the case where we combine all eight families. We train 10 decision trees which form the random forest based on 10 randomly selected subsets of the dataset and 10 random subsets of n-grams. The dataset comprises of 8 families all of which are fairly balanced in number. The random forest essentially turns this generic problem to a combination of specific problems and hence the results are stronger than the other cases.

The results for experiments conducted using 3-grams and 4-grams as features were similar. Refer to Appendix B for the results of those experiments.

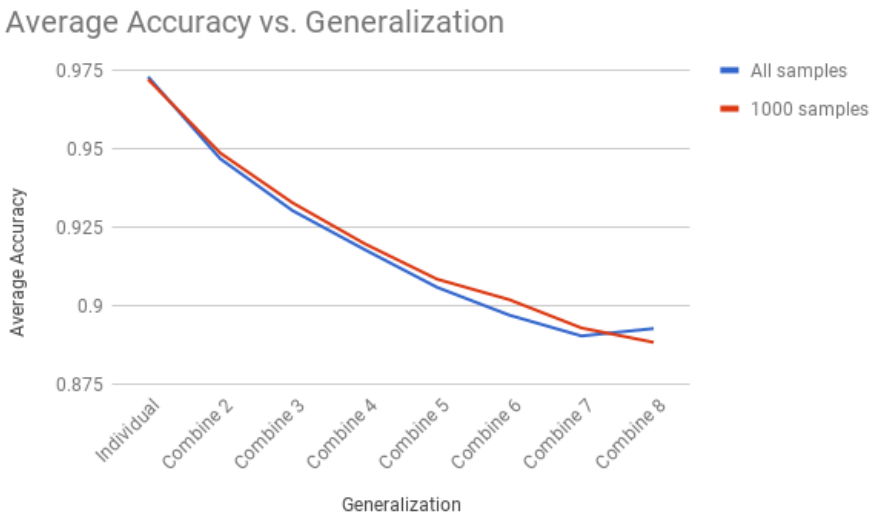


Figure 9: Random Forests Results - Average accuracy for generic models

4.6 Summary of Results

Figure 10 depicts the variation in average accuracy with increasing generalization of models for all the four techniques. The graphs shows that random forest is clearly the strongest technique followed closely by k-NN. SVM is better than χ^2 test for

detecting individual malware families but its accuracy drops steeply for more generic models. All of these techniques show that detecting malware is more challenging when the dataset is more generic.

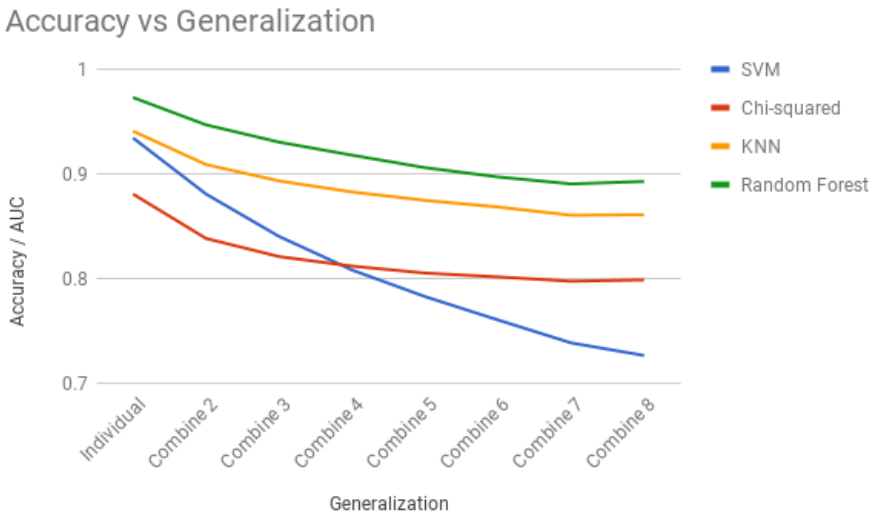


Figure 10: Summary of Results - Average accuracy for generic models

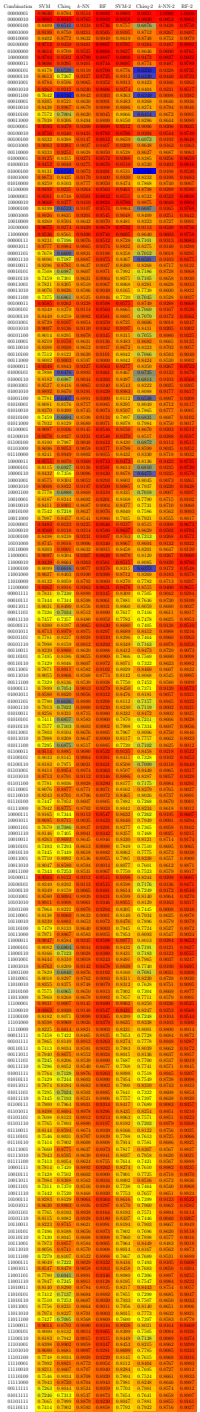
Table 9 summarizes all the results in a heatmap. The combinations of malware families are represented as a 8 bit mask where each family is assigned a bit. If the bit is set, it signifies the presence of the malware family in the experiment. Table 8 provides an index to read the heatmap.

The heatmap is arranged in ascending order of number of families used. Looking at the colors, we see towards the top, the accuracies are higher and they keep decreasing as we move towards the bottom where the number of families is more.

Table 8: Index for heatmap

Malware Family	Bit Number
Gatak	0
Kelihos	1
Lollipop	2
Obfuscator	3
Ramnit	4
Winwebsec	5
Zbot	6
Zeroaccess	7

Table 9: Heatmap of all results



CHAPTER 5

Conclusions and Future Work

5.1 Conclusion

We used four different machine learning techniques — support vector machines (SVM), χ^2 test, k-NN, and random forests using n-grams as features to evaluate the effectiveness of generic malware models. Experiments were conducted on all possible combinations of the seven malware families using all these techniques. We conducted experiments with balanced datasets where each family had 1000 samples and with imbalanced datasets where each family had different number of samples. We also experimented with different values of n i.e. we performed these experiments using 2-grams, 3-grams and 4-grams.

The SVM model performed well on individual malware families. However, the effectiveness of the SVM fades quickly when the malware set is more generic. In the most generic case, the SVM model could only manage a balanced accuracy of 73%. χ^2 was not as effective as SVM for the individual families but scaled better for the more generic tests. k-NN and random forests were both good at classifying individual families and their accuracy curves didn't dip as much as other techniques.

To summarize, SVM, χ^2, k-NN and random forests all performed well for individual malware families. The accuracy for all of these techniques dropped significantly when the models were more generic. Some of these techniques did better than others in the more generic cases. The random forest specifically was the strongest classifier with an accuracy of 88% for the most generic model.

5.2 Future Work

In this research we've observed the degradation of efficiency of malware detection techniques when the malware set is made more generic. We have also observed that techniques like random forests are less prone to being affected by generalization.

Additional features like opcodes and techniques like hidden Markov Models (HMM) could be explored in future. Previous research [11] has shown that dynamic features like API calls are strong features when used with HMM. It should be interesting to see similar experiments be conducted using API calls as features. An analogous set of experiments could also be performed on opcodes to see how they fare with generic models.

The random forest based classifier came closest to effectively detecting generic malware. This can be attributed to the double bagging built into the algorithm. It would be interesting to use bagging on the other techniques used in this research and compare their performance.

In this research we used all kinds of malware families with different kinds of behavior like Botnets, Trojans, Worms, Adware. A similar set of experiments with malware families of similar kinds should give an interesting insight on whether the behavior of malware affects their detection. It would also answer the question whether combining similar malware families is better than combining random families.

One of the challenges faced during this research was finding labeled datasets. There are a lot of malware datasets available but they contain all kinds of malware. It would be useful to develop a technique to cluster these malware into families without knowing anything about them. Such a mechanism would facilitate more of such experiments at a larger scale.

LIST OF REFERENCES

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

[2] "Symantec internet security threat report," https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf.

[3] M. Stamp, "A revealing introduction to hidden Markov models," https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf, 2004.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[5] 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. [Online]. Available: http://dx.doi.org/10.1007/s11416-006-0027-8

[6] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16 – 29, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1363412709000041

[7] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, ser. CSI-KDD '09, New York, NY, USA, 2009, pp. 23–31. [Online]. Available: http://doi.acm.org/10.1145/1599272.1599278

[8] 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 (ICIEA)*, June 2013, pp. 777–782.

[9] E. Raff *et al.*, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, pp. 1–20, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11416-016-0283-1

[10] J. Aycock, *Computer Viruses and Malware*, ser. Advances in Information Security. Springer US, 2006. [Online]. Available: https://books.google.com/books?id=xnW-qvk1gzkC

[11] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017. [Online]. Available: https://doi.org/10.1007/s11416-015-0261-z

[12] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006. [Online]. Available: https://doi.org/10.1007/s11416-006-0028-7

[13] T. Singh, F. D. Troia, C. A. Visaggio, 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. [Online]. Available: https://doi.org/10.1007/s11416-015-0252-0

[14] 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. [Online]. Available: https://doi.org/10.1007/s11416-014-0215-x

[15] "Trojan:win32/gatak threat description - windows defender security intelligence," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan%3AWin32%2FGatak.

[16] "Win32/kelihos threat description - windows defender security intelligence," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32%2fKelihos.

[17] "Adware:win32/lollipop threat description - windows defender security intelligence," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop.

[18] "Virtool:win32/obfuscator.acy threat description - windows defender security intelligence," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY.

[19] "Win32/ramnit threat description - windows defender security intelligence," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Ramnit.

[20] "Microsoft malware protection center, winwebsec," https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec.

[21] "Symantec security response, zbot," http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99.

[22] "Symantec security response, trojan.zeroaccess," https://www.symantec.com/security_response/writeup.jsp?docid=2011-071314-0410-99.

[23] "Microsoft malware classification challenge (big 2015) | kaggle," https://www.kaggle.com/c/malware-classification/data.

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

[25] 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.

[26] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[27] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review / Revue Internationale de Statistique*, vol. 51, no. 1, pp. 59–72, 1983. [Online]. Available: http://www.jstor.org/stable/1402731

[28] K. Pearson, "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900. [Online]. Available: https://doi.org/10.1080/14786440009463897

APPENDIX A

Benign dataset

Table A.10: Benign Dataset

alg	dpnsvr	lpr	rdsaddin	spiisupd
append	dpvsetup	lsass	rdshost	spnpinst
arp	driverquery	magnify	recover	spoolsv
asr_fmt	drwatson	makecab	redir	sprestrt
asr_ldm	drwtsn32	mem	regedt32	stimon
asr_pfu	dumprep	migpwd	reg	subst
at	dvdplay	mmc	regini	svchost
atmadm	dvdupgrd	mmcperf	regsvr32	syncapp
attrib	dwwin	mnmsrvc	regwiz	sysedit
auditusr	dxdiag	mobsync	relog	syskey
autochk	edlin	mountvol	replace	sysocmgr
autoconv	esentutl	mplay32	reset	systeminfo
autofmt	eudcedit	mpnotify	rexec	systray
autolfn	eventcreate	mqbkup	RmActivate	taskkill
blastcln	eventtriggers	mqsvc	RmActivate_isv	tasklist
bootcfg	eventvwr	mqtgsvc	RmActivate_ssp	taskman
bootok	exe2bin	mrinfo	RmActivate_ssp_isv	taskmgr
bootvrfy	expand	mrt	route	tcmsetup
BrowserChoice	extrac32	mscdexnt	routemon	tcpsvcs
cacls	fastopen	msdtc	rsh	telnet
calc	fc	msfeedssync	rsm	tftp
charmap	find	msg	rsmsink	tlntadmn
chkdsk	findstr	mshearts	rsmui	tlntsess
chkntfs	finger	mshta	rsnotify	tlntsvr
cidaemon	fixmapi	msiexec	rsopprov	tourstart
cipher	FlashPlayerApp	mspaint	rspndr	tracerpt
cisvc	fltMc	msswchx	rsvp	tracert6
ckcnv	FontReg	mstinit	runas	tracert
cleanmgr	fontview	mstsc	rundll32	tscon
cliconfg	forcedos	napstat	runonce	tsdiscon
clipbrd	freecell	netsetup	rwinsta	tskill
clipsrv	fsquirt	netsh	savedump	tsshutdn
cmd	fsutil	netstat	scardsvr	tswbprxy
cmdl32	ftp	nlsfunc	sc	typeperf
cmmon32	gdi	ntvdm	schtasks	tzchange
cmstp	getmac	nw16	sdbinst	unlodctr
compact	gpresult	nwscript	secedit	upnpcont
comp	gpupdate	odbcad32	services	ups
conime	grpconv	pentnt	sessmgr	user
control	help	perfmon	sethc	userinit
convert	hostname	ping6	setup	wextract
cscript	ie4uinit	ping	setupn	wiaacmgr
csrss	ieudinit	pintool	setupold	winchat
ctfmon	iexpress	powercfg	setver	winfxdocobj
dcomcnfg	imapi	print	sfc	winhlp32
ddeshare	ipconfig	progman	shadow	winlogon
debug	ipsec6	proquota	share	winmine
defrag	ipv6	proxycfg	shmgrate	winmsd
dfrgfat	ipxroute	qappsrv	shrpubw	winspool
dfrgntfs	krnl386	qfecheck	shutdown	winver
diantz	label	qprocess	sigverif	wksprt
diskpart	lights	qwinsta	skeys	wpabaln
diskperf	lnkstub	rasautou	smbinst	wpnpinst
dllhost	locator	rasdial	smlogsvc	write
dllhst3g	lodctr	rasphone	smss	wscntfy
dmadmin	logagent	rcimlby	sndrec32	wscript
dmremote	logman	rcp	sndvol32	wuauclt1
doskey	logoff	rdpclip	sol	wuauclt
dosx	logonui	rdpinit	sort	wupdmgr
dplaysvr	lpq	rdpshell	spider	xcopy

APPENDIX B

Additional Results

B.1 Results for 3-grams

Results for n=3 are presented below

Table B.11: SVM Accuracy for individual families using 3-grams

Malware Family	Balanced Accuracy
Gatak	0.9627
Kelihos	0.9832
Lollipop	0.8861
Obfuscator	0.8787
Ramnit	0.7568
Winwebsec	0.9678
Zbot	0.9470
Zeroaccess	0.9760

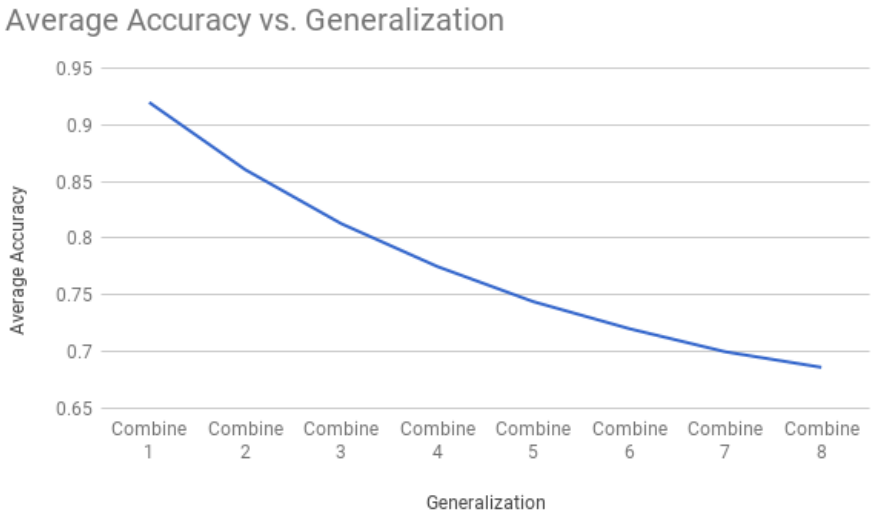


Figure B.11: SVM Results - Average accuracy for generic models using 3-grams

Table B.12: χ^2 test - AUC ROC for individual families using 3-grams

Malware Family	AUC ROC
Gatak	0.9696
Kelihos	0.9797
Lollipop	0.8306
Obfuscator	0.7899
Ramnit	0.8012
Winwebsec	0.9605
Zbot	0.8280
Zeroaccess	0.8504

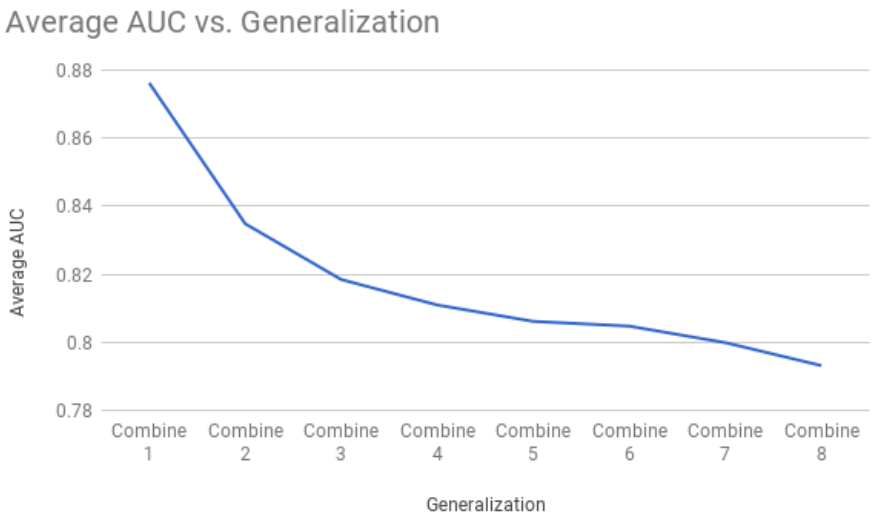


Figure B.12: χ^2 Results - Average AUC for generic models using 3-grams

Table B.13: k-NN Accuracy for individual families using 3-grams

Malware Family	Balanced Accuracy
Gatak	0.9639
Kelihos	0.9765
Lollipop	0.9520
Obfuscator	0.8816
Ramnit	0.7919
Winwebsec	0.9598
Zbot	0.9411
Zeroaccess	0.9716

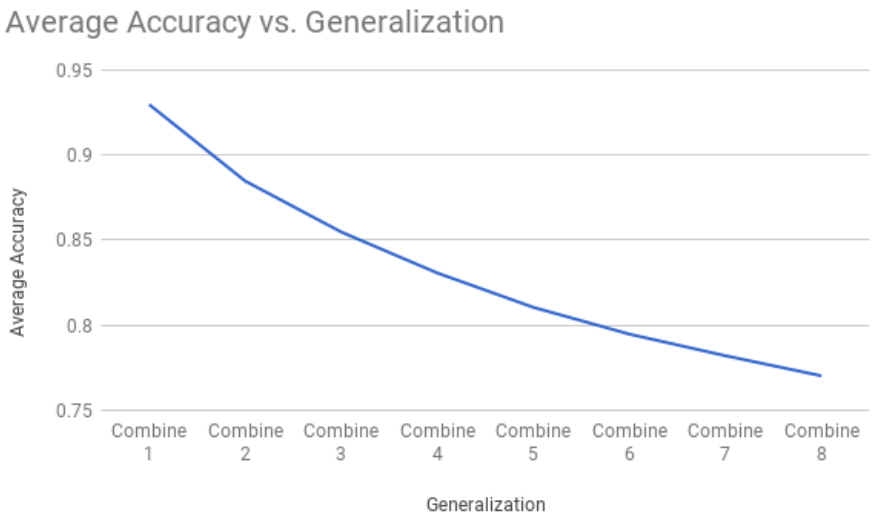


Figure B.13: k-NN Results - Average accuracy for generic models using 3-grams

Table B.14: Random Forests Accuracy for individual families using 3-grams

Malware Family	Balanced Accuracy
Gatak	0.9859
Kelihos	0.9913
Lollipop	0.9782
Obfuscator	0.9336
Ramnit	0.8653
Winwebsec	0.9863
Zbot	0.9568
Zeroaccess	0.9899

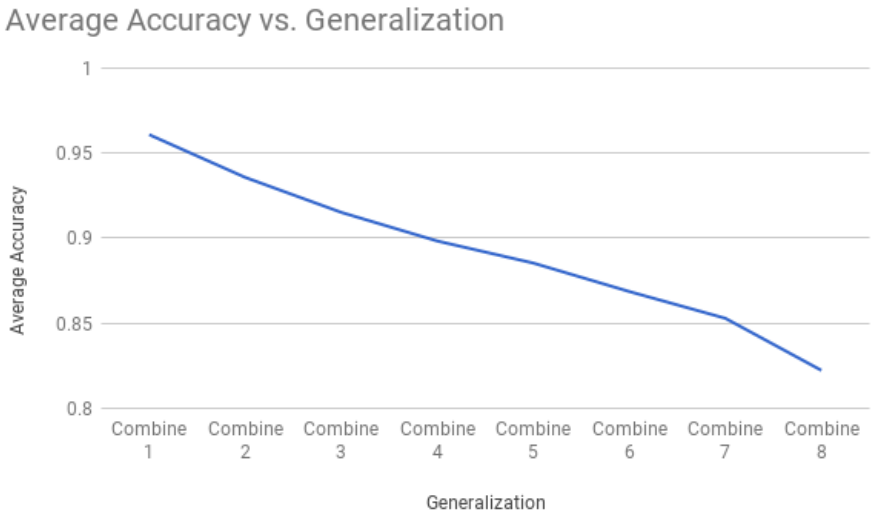


Figure B.14: Random Forests Results - Average accuracy for generic models using 3-grams

B.2 Results for 4-grams

Results for n=4 are presented below

Table B.15: SVM Accuracy for individual families using 4-grams

Malware Family	Balanced Accuracy
Gatak	0.9665
Kelihos	0.9891
Lollipop	0.8607
Obfuscator	0.8236
Ramnit	0.6568
Winwebsec	0.9376
Zbot	0.9247
Zeroaccess	0.9881

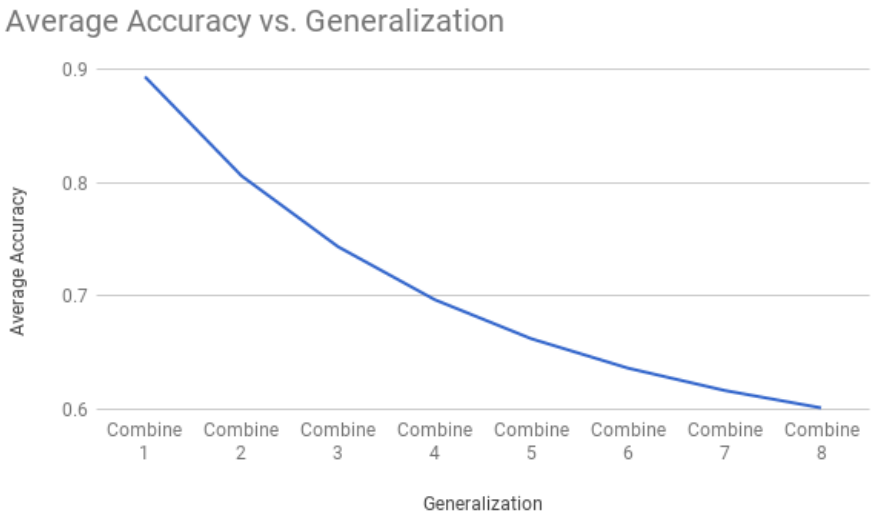


Figure B.15: SVM Results - Average accuracy for generic models using 4-grams

Table B.16: χ^2 test - AUC ROC for individual families using 4-grams

Malware Family	AUC ROC
Gatak	0.9155
Kelihos	0.9501
Lollipop	0.6068
Obfuscator	0.7283
Ramnit	0.7139
Winwebsec	0.8225
Zbot	0.8226
Zeroaccess	0.9179

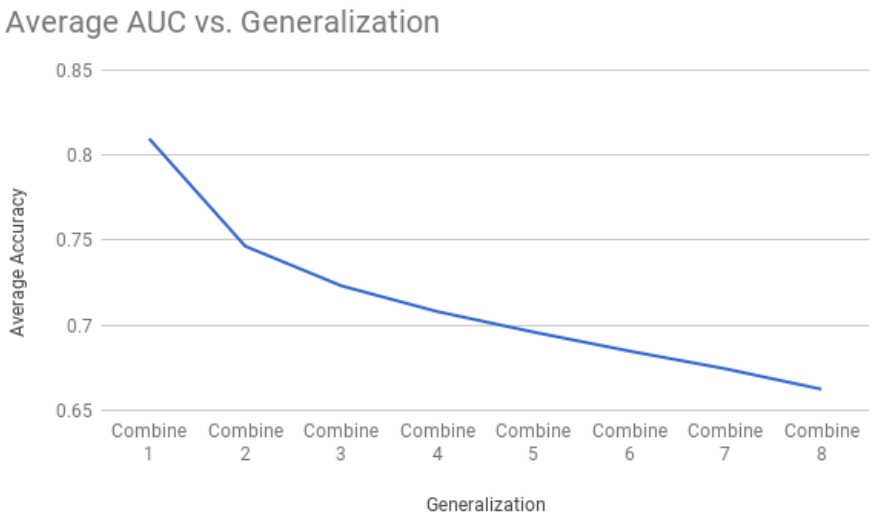


Figure B.16: χ^2 Results - Average AUC for generic models using 4-grams

Table B.17: k-NN Accuracy for individual families using 4-grams

Malware Family	Balanced Accuracy
Gatak	0.9699
Kelihos	0.9898
Lollipop	0.8960
Obfuscator	0.8691
Ramnit	0.7446
Winwebsec	0.9497
Zbot	0.9473
Zeroaccess	0.9831

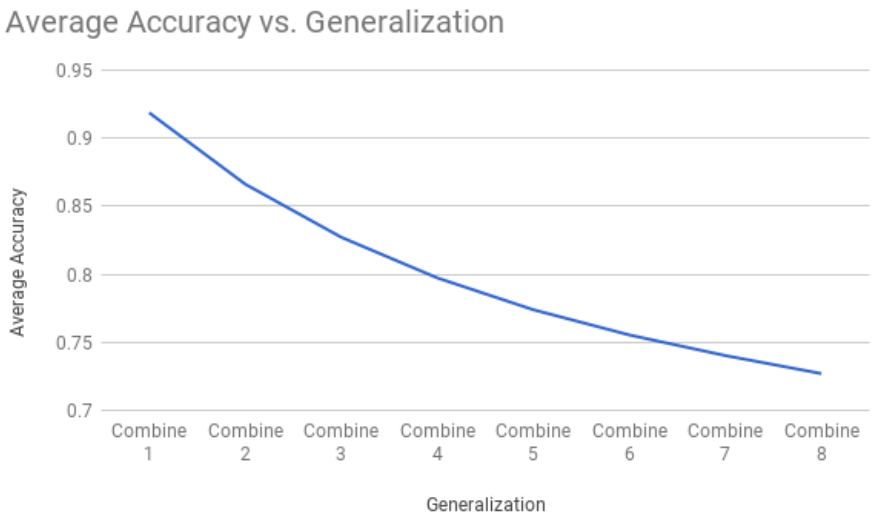


Figure B.17: k-NN Results - Average accuracy for generic models using 4-grams

Table B.18: Random Forests Accuracy for individual families using 4-grams

Malware Family	Balanced Accuracy
Gatak	0.9884
Kelihos	0.9975
Lollipop	0.9618
Obfuscator	0.9382
Ramnit	0.8875
Winwebsec	0.9864
Zbot	0.9718
Zeroaccess	0.9928

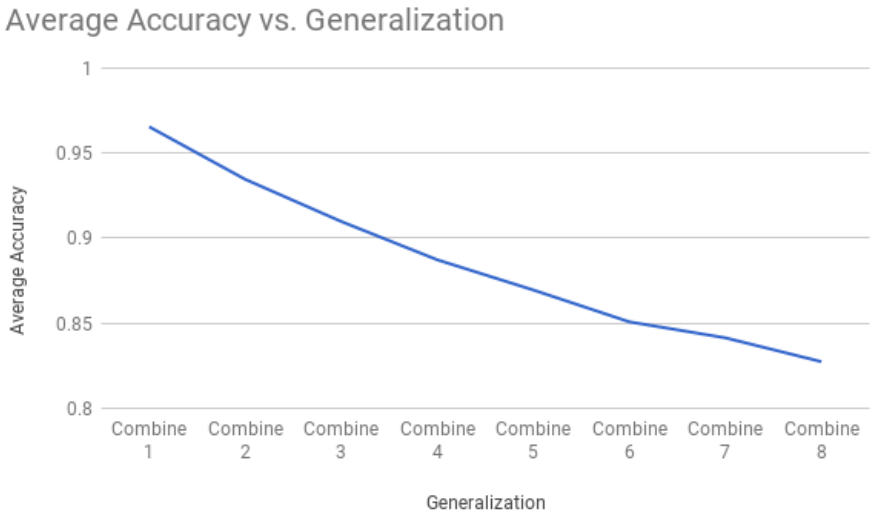


Figure B.18: Random Forests Results - Average accuracy for generic models using 4-grams