Malware Target Recognition via Static Heuristics[☆]

T. Dube[a,*], R. Raines[a], G. Peterson[a], K. Bauer[a], M. Grimaila[a], S. Rogers[b]

[a] *Air Force Institute of Technology, Wright-Patterson AFB, OH, USA, 45433-7765*
[b] *Sensors and Information Directorates, Air Force Research Laboratory, Wright-Patterson AFB, OH, USA, 45433-7321*

Abstract

Organizations increasingly rely on the confidentiality, integrity and availability of their information and communications technologies to conduct effective business operations while maintaining their competitive edge. Exploitation of these networks via the introduction of undetected malware ultimately degrades their competitive edge, while taking advantage of limited network visibility and the high cost of analyzing massive numbers of programs. This article introduces the novel Malware Target Recognition (MaTR) system which combines the decision tree machine learning algorithm with static heuristic features for malware detection. By focusing on contextually important static heuristic features, this research demonstrates superior detection results. Experimental results on large sample datasets demonstrate near ideal malware detection performance (99.9+% accuracy) with low false positive (8.73e-4) and false negative rates (8.03e-4) at the same point on the performance curve. Test results against a set of publicly unknown malware, including potential advanced competitor tools, show MaTR's superior detection rate (99%) versus the union of detections from three commercial antivirus products (60%). The resulting model is a fine granularity sensor with potential to dramatically augment cyberspace situation awareness.

Keywords: malware detection, information assurance, decision trees

[☆] Patent pending.

[*] Corresponding author, phone (937) 255-3636 x4690, FAX (937) 904-7979, e-mail: thomas.dube@afit.edu

1. Introduction

Malware heuristic analysis techniques generally fall into two distinct categories: static and dynamic [1]. Static heuristics generally use non-runtime indicators [1], such as structural anomalies, program disassembly and n-grams [2, 3, 4, 5, 6]. Alternatively, dynamic heuristics employ runtime indicators [1] normally obtained in virtual environments, such as commercial sandbox applications [7, 8, 9] or emulation capabilities of antivirus products [1, 10].

Despite the success that static heuristics enjoyed during the 1990s [1], today's research and commercial antivirus products heavily favor dynamic heuristics [1, 10, 11, 12, 13, 14]. Antivirus companies use a hybrid of static and dynamic heuristics in their commercial products [1, 10]. Most static heuristics require a pristine disassembly, which is difficult to achieve [15]. Dynamic heuristics avoid this limitation [15], because they do not require disassembly, but rather observe program execution in a restricted environment for a specified observation period. Observing program behavior requires all program dependencies to be present [1], which is a stronger requirement for dynamic heuristics than static heuristics. The program may not successfully execute in the test environment when a required runtime library is absent.

Dynamic heuristic methods are generally slower than static methods [10], because they require an observation duration and emulation overhead. Their performance makes them operationally infeasible to test tens of thousands of unique programs on a single system in tactical situations. Dynamic heuristic analysis is also incomplete [16], because no guarantee of observing malicious activity within the observation period exists. Many malware samples require a trigger condition [16] to demonstrate their malicious behavior. For example, the Michelangelo virus [17] only executes its payload on March 6, the anniversary of his birth.

This research extends current malware detection research in three important ways. First, Malware Target Recognition (MaTR) demonstrates the utility of using only anomaly and structural static heuristics for robust malware detection, in contrast to previous research using the same sources of information [2]. Second, this work also achieves a significant performance improvement over other static heuristic malware detection research [2, 3, 5, 18]. For fair comparison, MaTR competes against a retest of the Kolter and Maloof n-gram research [3], the best measures seen in similar work, on a larger dataset. Finally, a validation test against a publicly unknown malware set

shows MaTR's superior performance over an n-gram model and three commercial antivirus products.

The following sections describe related research, MaTR and another static heuristic detection methodology, and hypotheses tested. The next topics covered are the experimental comparison of MaTR with a repeated experiment from other researchers and results illustrating MaTR's performance advantage against a suite of commercial antivirus products. Conclusions summarize this work and include brief synopses of limitations, potential impact, and future research.

2. Related Work

While malware detection is a popular research area, nearly all current efforts focus on dynamic heuristic analysis. In static heuristic analysis, many efforts require the successful static disassembly of programs, which is commonly augmented by dynamic methods. Currently, the scope of MaTR is strictly static heuristic analysis, explicitly restricted to features readily available by cursory, non-runtime inspection of a program. This section briefly describes related research in static heuristic analysis of malware.

2.1. Kephart, Tesauro and Arnold

IBM researchers Kephart, Tesauro and Arnold provide the seminal research in n-gram analysis of malware. These n-grams are byte sequences of length n that occur in the target, which theoretically represent program structural components and fragments of instructions and data. They examine the use of n-grams in automatic signature extraction [19] for malware variants as well as for generic detection [18, 20].

While searching for methods to automate signature extraction for new variants of known malware, Kephart, et al. discover the utility of n-grams for generic malware detection [19]. By determining the probability of finding specific n-grams in malicious and non-malicious programs, the authors fabricate a generic malware detection classifier.

Tesauro, et al. successfully use neural networks to detect boot sector viruses [18]. They manipulate the decision threshold boundary to increase the cost associated with false positives as they cite that a single false positive reading likely affects thousands of systems. Despite significant computational

and space constraints as well as a small sample size for training and validation, they achieve a false positive rate of less than 1% while detecting over 80% of unknown boot sector viruses.

They train the network with trigrams (3-byte strings) that undergo a novel feature selection process. Initially, they canvas the entire sample corpus for trigrams and eliminate all that are common to both the malicious and non-malicious sets. Moreover, they reduce the list of trigram features to the set where each malicious training sample contains at least four trigrams. This selection process leads to a three order of magnitude feature reduction.

Expanding on their previous work, Arnold and Tesauro incorporate a voting system on multiple trained neural networks [20]. By training multiple networks with distinct features not used in others, they effectively avoid the major pitfall associated with heuristic scanners, high false positive rates. Their assumption is that these disparately trained networks rarely produce identical false positives. Szor cites that the Arnold and Tesauro network research has such a low false positive rate that Symantec incorporated it into its antivirus product default scanning [1].

2.2. Schultz, et al.

Schultz, et al. make key contributions by testing three different sources of features to identify malware [2]. In their first approach, they examine information from the portable executable (PE) header as features, such as import libraries and the number of imported functions from those libraries, with Cohen's improved rule learning algorithm called RIPPER [21]. This method requires unpacking the samples before evaluation to reveal the true imports, but the authors do not refer to this difficult step. The second approach uses strings found in the binaries as features, which is again problematic without first unpacking.

The third method captures byte sequences expected to translate loosely to a representation of instructions, data, or both. Without first unpacking the binary, however, in the best case one would expect such byte sequences to most likely represent general program structure, unpacker stub instructions and offsets, unpacker data, or other global information. In the worst case, these byte strings may represent packed information, which is essentially indecipherable data.

The authors used both the string and byte sequence data with a naive and multi-naive Bayes classifiers. The results of the naive Bayes classifier with the string features is the most accurate classifier in their tests reaching

a detection rate of 97.43% with a false positive rate of 3.80%. The authors concede that encryption (and presumably packing) obfuscate strings present in the executables [2], but the solution they suggest makes bold assumptions. They indicate that an effective method of handling the packing case is to initially assume that a sample is malicious and then if strings are found in the program the classifier defaults back to the naive Bayes algorithm.

2.3. Kolter and Maloof

Kolter and Maloof (hereon KM) also made key contributions by examining the results of several classifiers on malware detection via a common text classification technique, n-grams [3]. Techniques they test include naive Bayes, support vector machines (SVM), decision trees (DT) and boosted variants of each. In their experiments, they evaluate the classifier performance by computing the area under a receiver operating characteristic (ROC) curve. Their boosted DT model achieved the best accuracy, a 95% confidence interval area under the ROC curve (AUC) of 0.9958 ± 0.0024. The authors describe the difficulty of identifying why the presence of some byte strings combined with the absence of other byte strings contributes to high performance classifiers. In their study, they use 1,971 non-malicious executables from Windows 2000, XP operating systems, and SourceForge (`http://sourceforge.net`). Their malware sample set comprised of 1,651 samples obtained from the VX Heavens website (`http://vx.netlux.org`) and MITRE Corporation.

Expanding on their detection work, KM apply their n-gram methods to the identification of malware payloads as well [5]. The extension of static heuristic methods to identify malware payload functionality is significant as nearly all research efforts to identify malware functionality employ dynamic heuristics. The authors examine three major malware payloads: mass mailers, backdoors, and viruses. To gather the requisite data, they examine reverse engineering analysis reports to determine functionality. They manage to find functionality information for 525 of the 1,651 malware samples they previously obtained from the VX Heavens website.

In their experiments, boosted J48 decision trees performs best with statistically significant differences over a few of the methods they test, achieving the highest AUC confidence intervals for identifying backdoors and viruses of 0.8704 ± 0.0161 and 0.9114 ± 0.0166 respectively. They find SVM slightly higher (again, no statistically significant difference) than boosted J48 in identifying mass mailers with an AUC confidence interval of 0.8986 ± 0.0145. While the payload identification rates are not spectacular, they demonstrate

the ability to determine malware functionality without the need for lengthy dynamic analysis or close manual inspection—a substantial capability given the sheer volume of unique programs in the world and the pressing need for organizations to defend themselves. An n-gram based classifier should exhibit excellent runtime performance, because each feature is simply a linear ($O(n)$) search result for the appropriate byte sequence.

3. Detection Methodology

This article describes the results of a set of comparison tests using the KM n-gram approach and MaTR. Both of these methods employ static heuristic features and use ensemble decision tree machine learning algorithms for classification decisions. These subsections present different methodologies for determining specific features used by their respective classifiers.

3.1. KM Approach

In reconstructing the KM experiment, this research uses their described methodology [3] to generate n-grams and employs their identified length of $n = 4$ with a 1-byte sliding window. The KM methodology requires the extractions of 4-grams from all samples with respect to true class membership (true "clean" or "dirty" labels assigned during sample collection). They then determine a final feature selection of 500 n-grams for training and testing based on the highest n-gram information gains as computed by the following formula:

$$IG(j) = \sum_{v_j \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_j, C_i) \log \frac{P(v_j, C_i)}{P(v_j)P(C_i)}, \tag{1}$$

where C_i is the ith class (of the true sample labeling) and v_j indicates the presence or absence of the jth n-gram. The prior and conditional probabilities are self-explanatory. They treat the presence of an indicated n-gram as a Boolean feature to their boosted decision tree classifier.

3.2. MaTR Approach

The MaTR system uses a straightforward process for detecting malware using only a program's high-level structural data. While many researchers and commercial companies use this same structural data, none rely exclusively on this source of data and achieve the performance levels of MaTR.

Figure 1 shows the inputs and outputs of MaTR and illustrates its internal process. Inputs to MaTR are executable files, such as portable executable (PE) files common in the Microsoft Windows operating systems.

Although an open system, MaTR explicitly bounds the machine and human operator together within the overall system, a subtle yet significant distinction from other work that simply uses a computer to generate "answers". In MaTR's architecture, the operator becomes a critical component receiving and providing feedback to the rest of the system and eventually initiating a response action.

Recognizing the operator's role allows for a more robust network defense. The discoveries of certain malware payloads (an area of future work) require different responses. For instance, the standard antivirus software response to nearly any malware discovery is to clean or quarantine the malware sample. If the malware has a "password stealer" [22] capability, cleaning or quarantining is an insufficient response action by itself, because they only remedy further and not current malware losses. If the architecture presents this information to the operator, the rational human response should include cleaning or quarantining, but also mandatory password changes and an investigation into potential loss of data confidentiality. MaTR presents the pertinent threat information to operators enabling them to initiate the most appropriate response actions.

Limiting features to contextually significant information is a requirement to maximize potential feedback with a human operator. One can visualize this benefit when considering the comprehension difficulty for a human faced with the resulting decision process of an n-gram solution or the lack of decision making information provided by a commercial antivirus product that only provides the final result. The co-alignment of the human operator and the machine within MaTR allows for critical and constructive feedback, which remains an area for continued work.

The "Data Pre-processing" stage allows for any steps required before feature extraction and subsequent classifications. Data pre-processing actions include discovery of valid executable files. Other actions include pre-filtering known malware and known non-malware, decrypting data, and data sharing with other sensor systems.

During "Feature Extraction", the system parses the input file to find the predetermined data inputs for the subsequent classifiers. Features (described later) are restricted to the input file's high-level structural anomalies and raw structure information. "Feature Transformation" involves any action taken

on features before classification, such as bounding, mapping, projecting, etc. Examples of well known transformations include principal component analysis and factor analysis.

The "Detection Classifier Data" component represents the data for the trained classifier. For example, decision tree classifiers must correctly initialize a binary tree node structure with appropriate cut features to uniquely identify the specific feature to test, cut values and classification decisions for the leaf nodes of the decision tree.

The underlying decision tree classifier comprises the "Detection Classification" component. At this point, the classifier takes the transformed features and makes its classification decision based on its underlying algorithm. For example, in decision trees, the decision sequence begins at the root node and progresses down to a single leaf node where the classification decision is determined. "Detection Post-processing" allows for post-filtering before presenting preliminary results to the operator, triggering additional actions, result verification with other systems, or data fusion with additional sensors.

As MaTR does not rely on computations to determine the final feature set, it avoids the overhead of a resource-intensive feature selection step [3]. However, the KM method results in a simpler and more efficient feature extraction step. This results in a trade-off as MaTR's feature extraction process remains more complex throughout its life cycle.

3.2.1. Bagged Decision Tree Classifiers

The MaTR architecture employs bagged decision trees as its classifier based its performance on previous experiment results. The decision tree is a machine learning classifier with a tree data structure. Classification decisions are the result of traversing from the tree root to a leaf node. Each non-leaf node employs a split variable and split value to determine the path of traversal to a leaf node, where each tree makes a final class assignment. Each leaf bases the assignment on prior probabilities from the remaining sample subpopulation at that leaf established during training.

The MaTR implementation tested uses the MATLAB bagged decision tree implementation `TreeBagger` [23]. Training decision trees involves determining the proper feature and value for each node to split the training set into subpopulations with lower node impurity (a measure of the subpopulation consisting of same class labels). For each decision split, `TreeBagger` by default randomly selects $\sqrt{n}$ features of n total features as candidates for the split variable and assesses them using one of the following impurity

functions: Gini's diversity index, the twoing rule and the maximum deviance reduction [23].

Bagging, or bootstrap aggregation, is an augmentation method of the performance of a single tree by generating a tree ensemble with each tree based on different bootstrap samplings. During training, the selection of n samples from the training set (of size n) with replacement constitutes a bootstrap sampling. The resulting classifier uses a majority vote of the individual trees in the ensemble.

The MaTR model tested uses an ensemble of 25 trees with default parameters. By default, `TreeBagger` considers $\sqrt{n}$ random features for each cut variable, allows a minimum of one observation per leaf node and employs the Gini's diversity index as an impurity measure [23]. Other defaults include no pruning and using equal misclassification costs [23].

3.2.2. MaTR Features

Perhaps the greatest distinction between MaTR and other commercial and research products is its feature set. MaTR achieves high detection performance while restricting its features exclusively to high-level program structural anomalies and general structural data. Instead of following a mathematical model to determine features, MaTR utilizes features commonly used by analysts [1, 24, 25] when examining samples to determine if they are indeed malicious. Rafiq and Mao found that malware routinely contains structural anomalies (78%), while non-malware does not (5%) [24].

The term "high-level" structural data refers to the basic structural format that the operating system loader uses when loading an executable program into memory before runtime and higher level information, such as common file attributes (e.g., name, path, file size, attributes, etc.). The sources for the structural anomalies come from a number of publications and personal observations of program structure. Combining expert experience with program structural information capitalizes on analysts experience while allowing for identification of additional anomalous feature combinations.

As analysts examine samples, their previous experiences contribute to a prior knowledge of analysis technique effectiveness and past observations. Significant observations useful for confirming malice are anomalies primarily seen in malware. Routinely, analysts combine available anomaly information with structural information to either confirm their suspicion or look for additional anomalies. For instance, if the visible program disassembly is insufficiently small to provide any significant advertised function, the analyst

may suspect that the program is packed. Many popular packers dedicate a program section for unpacking, but the section must allow reading and executing (as it will soon contain code), but it must also allow writing to unpack the obfuscated code before attempting to execute it. Analysts confirm these section permissions, or characteristics, by examining structural information for yet another anomaly.

Currently, MaTR utilizes over 100 static heuristic features based on structural anomalies and structural information itself. Many of MaTR's features are integral, unlike the KM method which uses exclusively Boolean features. MaTR does not attempt to generate an instruction disassembly due to the difficulty of validating its correctness [15] nor does MaTR use instruction sequence signatures as commercial antivirus programs use [10].

Structural anomalies are generally logical operations on program header information or file areas pointed to by header information. Classes of structural anomalies include: section names [1, 24, 25], section characteristics [1, 24, 25], entry point [1, 25], imports [2, 24, 25], exports [25], and alignment [1]. Structure information, included to enable classifiers to identify additional anomalous combinations, comes directly from the PE headers, such as the `IMAGE_FILE_HEADER` and the `IMAGE_OPTIONAL_HEADER` [26]. A description of some of the more popular anomaly features follows.

Non-standard section names. Several researchers [1, 24, 25] also identify the presence of a non-standard section name as anomalous. Microsoft [26] defines several standard section names for PEs and many compilers adopt this standard. This standardization has led to an overwhelming majority of non-malware containing only standard section names. According to Rafiq and Mao [24], only 3% of non-malware use unconventional section names, while 80% of malware samples use non-standard names.

Non-standard section characteristics. Many researchers [1, 24, 25] identify non-standard section characteristics as an anomaly. If a code section has read, execute and *write* characteristics instead of the normal read and execute characteristics, it immediately raises analysts' suspicions. Normally, the program uses sections with these permissions to unpack obfuscated code before attempting to execute it. This particular anomaly is common in malware, because packing is a common malware armoring technique [1].

Entry points. A program entry point that points to a section not marked as containing code is anomalous [25]. Szor states that a program whose entry

point does not point to the code section (`.text` for default compiling) is another entry point anomaly [1]. Packers commonly adjust the entry point to point to an additional code section to start the unpacking process.

Imports. Inclusion of information regarding import libraries and functions is common among malware research [1, 24, 25]. Common features include the numbers of import libraries and functions. Executables with a low number of imported functions are suspicious [25], because programmers normally provide program utility by importing functions, such as I/O, encryption or complex math.

Exports. Treadwell and Zhou also identify dynamically-linked libraries that export no functions as anomalous [25]. Since the purpose of a dynamically-linked library is to provide functionality to other programs via exported functions, the absence of exported functions is surely suspicious.

4. Theory

The presented background information concerning strategic employment of malware by competitors to gain advantage clearly demonstrates motivation for such—albeit illegal—activities. If a major defense contractor builds a weapon system with specific capabilities, an intimate knowledge of those capabilities and engineering designs to achieve them may allow a competitor to build a superior system. The obtained information enables the competitor to use the victim company's intellectual property as a baseline for their system.

The first major theoretical hypothesis MaTR tests is that static analysis techniques are inadequate to detect modern malware. Specifically, given MaTR's and the KM methodologies already described "shallow" investigation of sample executables, the following experiments must demonstrate that classifiers built with this level of information are adequate for practical applications.

Another hypothesis MaTR addresses is the assumption that commercial antivirus systems alone are inadequate to defend against advanced, competitive threats. Occasionally, information assurance practitioners have advocated using multiple commercial antivirus products to address these same threats. Commercial antivirus products likely are inadequate for organization strategic defense, because of their availability to the attacker for testing against prior to conducting information operations.

5. Experiments on Known Malware

These subsections describe the sources of samples for the subsequent experiments and the experimental designs and the measures of effectiveness used to test the above theories. The experimental designs focuses on establishing an assessment between MaTR and the KM n-gram research. The measures of effectiveness chosen allow for full comparison to other work.

The remaining subsections describe both observations and experimental results. These observations are useful as they provide valuable insight not available in the experimental results. For instance, previous n-gram research presents little evidence of what n-grams truly represent, which keeps the semantics of this method shrouded in mystery.

5.1. Data Collection

The following experiments examine only 32-bit portable executable (PE) samples obtained from well known sources. All "clean" samples come from harvesting of clean installs of Microsoft Windows XP, Vista, and Windows 7, while the malware samples come from an updated download of the VX Heavens dataset [27]. Specifically, the malware, or "dirty", samples are Trojans, worms, and viruses types as identified by the antivirus label assigned to them. Extractions of PEs from these sources yields 25,195 clean and 31,193 dirty samples for a total of 56,388 samples. These tests do not currently use samples from SourceForge as in [3].

KM use `hexdump` to capture n-grams in their experiments [3, 5], but `hexdump` has documented side effects when dumping hexadecimal representations when using format strings to control output [28]. Specifically, when the format string calls for 4 bytes and the sample has less than 4 bytes remaining, `hexdump` zero-pads the output. To avoid this error, a custom program extracts the n-grams for this experiment given the requirements from KM [3, 5].

5.2. Experimental Design

This experiment is a side-by-side comparison of leading static analysis malware detection techniques, specifically MaTR and the previous KM n-gram research [3, 5]. For consistency with prior research, these tests both adopt a standard experimental design using stratified, ten-fold cross validation. Each disjoint fold contains roughly the same number of samples from

malware and non-malware sets. During each run, a different fold functions as the test set while the remaining folds comprise the training set.

Each fold requires a determination of the top 500 n-grams specific to that fold's training set for the KM technique. Classifiers train on only the samples from a fold's training set and test results come from application of the trained classifier to the fold's test set. MaTR and the KM retest use identical folds.

These experiments use decision trees since both KM [3] and Dube, et al. [29, 30] both found these the best performing for this problem set in previous work. Both techniques use the ensemble `TreeBagger` classifier from MATLAB [31] with default parameters and 25 trees. The major differences in the KM retest and the original work [3] are the use of MATLAB instead of the `J48` decision tree implementation in WEKA [32] and the larger sampling.

5.3. Measures of Effectiveness

In order to fully compare these experimental results to other published research, this effort includes the results of a variety of measures commonly used in this research area. KM report ROC AUC results with confidence intervals [3]. Schultz, et al. report accuracy with FPR, but also provide confusion matrix data, which allows for calculating FNR [2]. Tesauro, et al. include general values of accuracy and FPR [18]. ROC measures use pooled averages across folds as described by Maloof [33]. Computed confidence intervals result from studentized bootstrapping with the sampling defined by the ten cross-validation folds.

Comparison of performance data from the KM retest with the original work serves as a validation measure for the experiment. Given the high accuracy KM achieve in their tests, any significant lesser performance in the retest would require a full analysis of variance using the `J48` solution in WEKA [32] and substantially smaller sample sizes to determine the the source of variation. KM claim that larger sample sizes would increase performance [3].

5.4. Data Collection Observations

Using the described KM parameters for generating n-grams yields a mean of 2,103,005,388 unique n-grams across training sets. Given that $n = 4$, this results in a 49% saturation of the possible 4-gram space. KM observe a saturation rate of only 6% in their large dataset in their original experiment.

Determining the set of n-grams using the KM method requires extensive "computational overhead" as they attest [3]. The datasets become too large

to store in memory and as a result, calculations must resort to heavy disk-utilization with deliberate runtime performance optimization. The number of expected unique n-grams is a critical implementation factor, as it is key in determining how best to partition the n-gram data space.

In this experiment, the KM n-gram generation technique generates a mean of 2,103,005,388 unique n-grams across training sets. This results in a larger saturation level of $2,103,005,388/2^{32} = 49\%$ compared to the saturation level of 6% from the KM research [3]. While this saturation level causes complications for determining the top n-grams to select, it does not impede the KM model classification performance, because the saturation of the n-gram space does not affect final model decisions which occur in the leaves of the decision trees. Theoretically, their model uses 500 Boolean features which yields $2^{500} = 3.27e150$ potential leaf combinations given the decision tree classifier.

Figure 2 is a plot showing the number of unique n-grams growing as the number of files parsed increases. Unfortunately, the two have a clearly linear relationship for the range tested with a strong Pearson's correlation of 0.9950. The larger sample sizes forces calculations to predominantly disk-based solutions.

5.5. Selected n*-gram Observations*

The KM method selects a total of only 505 unique n-grams to use as features across all ten folds making fold selections quite consistent. Table 1 shows the top seven n-grams for all folds. The primary difference of the remaining n-grams across folds is their order.

One observation about this partial listing is that the selected n-grams appear to focus on capturing specific byte sequences peculiar to each class. For instance, the first n-gram `0x00560001` is a 1-byte shift from the second n-gram chosen `0x56000100`. This pattern propagates through the selections with potentially longer byte sequences daisy-chained together.

A second observation is the prevalence of zero bytes (`0x00`) throughout the selections. Nearly 44% of all selected n-gram bytes are zero bytes. Closer examination of the zero bytes reveals a potential pattern of UNICODE character representations, zero bytes followed by non-zero bytes. This pattern is visible in 79% of all n-grams selected.

KM describe the difficulty in validating why n-grams work for classifying PEs [3]. As an example, they found a mysterious n-gram (`0x0000000a`) in their studies [3, 5], which they can not attribute as being code, data,

or structure. This specific n-gram `0x0000000a` is found in a comparable percentage of samples in the expanded malware set from VX Heavens as KM cite, but the same n-gram also appears in 83% of the non-malware set and the information gain feature selection algorithm never ranks it in the top 500 for any folds. Why KM focus on this particular n-gram is uncertain as they do not specify whether their calculations lead to its inclusion in the top 500 n-grams.

MaTR avoids some of the validation problem by using only contextually important information as features as described in Section 3.2.2. Using common anomalies and irrefutable structural information that analysts routinely use in making their assessments provides strong validation of MaTR's results. As a result, an analyst can confirm its decisions based on meaningful observations. The major difficulty with interpreting MaTR's decisions is the complexity of following the logical steps of an ensemble method—even the apparently intuitive decision trees.

5.6. Model Observations

The resulting classifiers from the original KM research are ensembles of small trees [3], averaging 90 nodes. In the KM retest, the tree sizes are much larger averaging 2,824 nodes per tree. Given the 49% saturation of the 4-gram space and the much larger sampling in the retest, the trees likely had to grow substantially to minimize impurity at the leaf nodes. MaTR averages 354 nodes per tree in these tests, which is approximately 3 times the tree size observed in previous MaTR research with smaller datasets [30].

The trees in the KM retest have less efficient representations as all features are Boolean (nominal, "present" or "absent"), which forces trees to grow significantly larger to accommodate the increased saturation of the n-gram space. When training decision tree classifiers with larger sample sets, the impurity levels of nodes near the root remain relatively high after new splits as the sample subpopulations are more diverse (underfitting). In order to improve classification accuracy effectively, the decision tree must grow in size.

The simpler tree representations of MaTR are likely due to the expressive power of augmenting nominal features with continuous interval and ratio features or an inherent difference in feature saliency. Interval and ratio features often represent ranges of values and relational comparisons in addition to equality. Including such features results in a denser information representation for the decision tree.

5.7. Experimental Results

Figure 3 shows a magnification of the ROC curves for both MaTR and the KM n-gram retest. While both methods demonstrate excellent results, MaTR achieves the more ideal ROC curve as it tracks closer to the left and top sides, resulting in a mean AUC of 0.999914 for MaTR compared to 0.999173 for the KM retest. Furthermore, MaTR never exhibits a lower true positive rate (TPR) or a higher false positive rate (FPR) than the KM retest for any given threshold values tested for the ROC plot. While the resulting AUC performance difference is statistically significant, it is not necessarily practically significant as both methods are close to ideal.

Tables 2 and 3 are the resulting AUC and accuracy confidence intervals for MaTR, the KM retest, and past research. The AUC results for the KM retest are a statistically significant 0.34 % improvement from their original research [3, 5]. This observation is quite interesting considering the increased saturation of the possible n-gram space for this larger test, but the classifier adequately compensates by extending the length of branches to utilize more of the available combination space.

Although the confidence intervals for MaTR and the KM retest are close, MaTR demonstrates superior results that are statistically significant to both the KM original and the retest. This consistency may indicate a higher saliency value of structural and anomaly data for detecting malware than n-grams, which are typically used for text classification. However, both results strongly suggest that static heuristic methods remain viable for malware detection.

For comparison with other research, Table 3 includes the apparent accuracy statistics. MaTR's accuracy is significantly better than those for the KM retest as the confidence intervals do not overlap. Analysis of the additional measure now leads to practically significant results as the accuracy results of the KM retest are nearly a full percentage point below MaTR's results. The accuracy advantage of MaTR is an aggregate indicator of a significant impact on its operational utility. Discussion of this impact requires computation of FPR and FNR (addressed later).

The best finding from Schultz's work [2], the strings classifier, has a much lower mean accuracy, and they do not include any confidence interval to describe variability in their published research. The simplicity of defeating a classifier based solely on strings [2] was a key factor in the decision to not repeat their experiment or a similar variant.

Additionally, Schultz's best structure/anomaly result has a mean accuracy of 0.8936, which is substantially lower than MaTR's. This discrepancy is most likely attributed to the small sample sizes used in their work. They state in their published research [2] that they had a limited subset of 244 PEs (206 benign and 38 malicious).

Table 4 shows the mean confusion matrix elements across the ten folds for the experiment. In the table, TP, FP, TN and FN stand for the standard true and false positives and negatives. MaTR averages only 5 total misclassifications, while the KM retest has 57. Both results are impressive considering the number of samples tested.

The confusion matrix data provides the values to determine the FPR and false negative rate (FNR) as shown in Table 5. Again, Schultz, et al. do not report confidence interval data, but their reported FPR and FNR appear quite different than both MaTR and the KM retest results. Once again, the MaTR results for both FPR and FNR are significantly superior to those of the KM retest. Furthermore, the MaTR FPR and FNR is lower than the 1% and 15-20% respectively from Tesauro's work, while MaTR additionally detects forms of malware other than boot sector viruses.

While MaTR's accuracy is consistent with its AUC results, the KM retest reveals an unexplainably lower accuracy than one may anticipate. The resulting ROC curves previously shown in Figure 3 provide evidence that suggests a lower accuracy for the KM retest. The observed mean TPR and FPR are 9.94e-3 and 1.37e-2 respectively. The true negative rate (TNR) is then 9.86e-3 ($TNR = 1 - FPR$). The average of the TPR and TNR is 0.990, which is an estimate of the measured accuracy (0.989919) as the model has equal misclassification costs. Generating the estimate for MaTR, the estimate is 0.999 compared to the observed measure of 0.999166. As far as the n-gram AUC measure disparity, Lobo, et al. [34] describe how impractical operating regions can inflate the resulting measure. While MaTR and the KM retest have similar AUC measures, the ROC plot for MaTR in Figure 3 is nearly ideal as it extends to the upper leftmost corner. The n-gram method does not achieve TPR rates comparable to MaTR (0.995 TPR) with less than a FPR of 0.02. For the same TPR, MaTR's FPR is negligible.

Finally, these FPR and FNR results illuminate a significant operational utility advantage of MaTR's methodology versus KM's. Operationally, the FPR directly relates to additional analyst workload, which is a form of resource waste as the additional samples are all non-malicious. The FNR also has operational implications, because it describes the method's inability to

detect malicious samples. While neither a high FPR or a high FNR is desirable, arguably the FPR is most significant, because it has such cascading effects given the normal distortion of sampling from the non-malware and malware classes.

For example, a typical clean installation of an operating system and office productivity software normally yields approximately 10,000 unique PEs, and this number will continually increase during system deployment. An advanced competitor may only require 1 or 2 malware artifacts to conduct effective offensive information operations. Given this estimate of a best case scenario, a 0.1% FPR yields 10 additional non-malware samples for an analyst to examine in addition to any malware samples detected. If the FPR rate is higher, the factor for resource waste increases linearly. This example also illustrates the value of a low FNR, because a method with a high FNR may miss the small number of malware artifacts present on a system.

6. Experiments on Unknown Malware

This section describes the unique source of data for a validation experiment comparing MaTR and the KM retest as well as three major commercial antivirus products. Using publicly unknown malware samples in this validation test clearly demonstrates a major theme of this research, the extensibility of malware detection methodologies to a realistic, operational environment to detect currently unknown threats. The following subsections describe the experimental designs and the measures of effectiveness used to test the above theories. The experimental designs focuses on establishing an assessment between MaTR and the KM n-gram research as well as testing both of these research methods against commercial antivirus products.

6.1. Data Collection

The data source for this test is a set of 278 malware samples discovered by multiple anonymous organizations. Local organizational policies generally restrict distribution of any discovered samples—even to antivirus vendors for signature development. These organizations believe these samples potentially include custom malware employed by aggressive competitors giving this particular sample set high strategic value. The only samples used in this particular test are malware samples.

6.2. Experimental Design

This specific test compares the performance results of MaTR, the KM retest, and three major commercial antivirus vendor products on the unknown malware samples. The only measure recorded for this test is TPR, because of the lack of any negative class samples and the limited thresholding capabilities of the commercial antivirus products tested. In this case, the dataset is strictly a test set split into appropriate folds. No classifier training uses extracted features from any of the unknown malware set samples.

For MaTR and the KM retest, this test uses the highest accuracy (already trained) classifiers from the previous test results in Section 5. Due to the smaller test sampling, pilot studies showed relatively large confidence intervals when conducting only the 10-fold cross validation runs as in the previous tests. Accordingly, this test replicates the 10-fold cross validation runs 10 times using unique validation sets for each replication. Otherwise, this test uses the same methodology as the previously described tests in Section 5.

The commercial antivirus products use default installation settings and have current updated signatures at the time of this experiment. Product scan results against the entire unknown malware set yield product-specific sets of all signature and heuristic hits. The intersection between the sample sets associated with each fold and these sets indicates the number of correct detections, while the difference is the number of false negatives. Otherwise, antivirus product test measures and confidence interval computations are identical to MaTR and the KM retest.

6.3. Results and Discussion

Table 6 shows the performance results against the unknown malware samples. Both MaTR and KM retest results resemble their previous test performance, but exhibit performance drops of 1.4% and 4.5% respectively. The disparity in performance between these two techniques increases against the unknown malware set and the differences are statistically significant. None of the commercial antivirus products exceed 50% TPR on the unknown malware set, a clear indication of the need for detection tools similar to MaTR.

Given the antivirus product detection information, further description of the unknown malware dataset is possible. For instance, the union of all antivirus detections accounts for 60% of the 278 samples, which validates the findings from the organizations who discovered them, but this observation has another implication. An occasional proposition in information assurance

circles is the suggested employment of multiple antivirus products. Considering that a combination of three commercial antivirus products yields only a 60% detection rate on these samples, implies that the return on investment, especially for large volume enterprise license purchases, is limited.

Combining generic detection methods, such as MaTR and the KM method, with commercial antivirus products may simplify discovery of advanced, competitive threats. For instance, antivirus products detect high rates of common malware, but their performance drops dramatically against unknown malware. However, the difference between sets of detections from a commercial product and a generic detection method should contain primarily advanced threat samples.

7. Conclusions

Pattern recognition techniques can play a substantial role in malware detection especially in cyber situation awareness and mission assurance. In exceedingly complex networks, simplifying assessment of operational readiness is a significant improvement and leads to better risk management. MaTR's high confidence detection rate coupled with its low FPR enable an aggressive defense against adversary intrusion. Furthermore, its low FNR implies that MaTR does not often mistake malware for benign software, which is also highly encouraging.

MaTR's performance results are convincing evidence that static heuristic methods are still operationally viable for malware detection, even without detailed instruction disassembly information. MaTR also demonstrates a significant advancement over previously published static heuristic methods, even on research using similar features. Not only does MaTR have the highest accuracy and AUC, but also the lowest FPR and FNR. Furthermore, MaTR achieves superior results while using only contextually important observations as features.

The test results of unknown malware samples with MaTR, the KM retest, and commercial antivirus products demonstrate MaTR's suitability for detection of unknown malware in an operational environment. This set of unknown malware is a significant sampling of tools like those employed by advanced threats. While MaTR detects nearly 99% of the unknown samples, the commercial antivirus products combine to detect only 60%. One cannot overstress the strategic value of such an advantage.

Accurate detection of malware with a low FPR provides maximum efficiency for prioritizing malware analysis operations, specifically prioritization for more resource intensive dynamic analysis methods and human analysts. A combinatorial approach can significantly augment the effectiveness of either method alone, because the hybrid solution can more thoroughly assess likely targets first.

7.1. Impact

The high accuracy in generic malware detection provides a significant fine granularity capability advancement for cyber situation awareness within complete local organization control. Given the true positive rates of both MaTR and the KM retest versus current commercial antivirus products, a static heuristic malware detection method is a potentially "game changing" technology that can shift the cyber battlefield in overwhelming favor of the defenders. It also provides critical information to enable organizational leadership to consider available response options and future defense investments.

8. Vitae

Major Thomas Dube serves in the United States Air Force and is currently a Ph.D. candidate at the Air Force Institute of Technology. Most recently, he worked in the tactics and assessment squadrons at the former Air Force Information Operations Center at Lackland AFB, TX. His previous research examined the application of malware defenses for non-malicious software. He also developed near real-time simulation systems at the Air Force Research Laboratory.

[1] P. Szor, The Art of Computer Virus Research and Defense, Addison-Wesley, Indianapolis, IN, USA, 2005.

[2] M. Schultz, E. Eskin, E. Zadok, S. Stolfo, Data Mining Methods for Detection of New Malicious Executables, in: IEEE Symposium on Security and Privacy, IEEE, 2001, pp. 38–49.

[3] J. Kolter, M. Maloof, Learning to Detect Malicious Executables in the Wild, in: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2004, pp. 470–478.

[4] T. Abou-Assaleh, N. Cercone, V. Keselj, R. Sweidan, N-gram-based detection of new malicious code, in: Proceedings of the 28th Annual International Computer Software and Applications Conference, IEEE, New York, USA, 2004, pp. 41–42.

[5] J. Kolter, M. Maloof, Learning to Detect and Classify Malicious Executables in the Wild, Journal of Machine Learning Research 7 (2006) 2721–2744.

[6] O. Henchiri, N. Japkowicz, A Feature Selection and Evaluation Scheme for Computer Virus Detection, in: Proceedings of the Sixth International Conference on Data Mining, IEEE, 2006, pp. 891–895.

[7] Sunbelt Software, Sunbelt cwsandbox, accessed Sept. 18, 2009 (Sept. 2009).
URL `www.sunbeltsoftware.com/Developer/Sunbelt-CWSandbox/`

[8] Norman ASA, Norman sandbox, accessed Sept. 18, 2009 (Sept. 2009).
URL `www.norman.com/technology/norman\_sandbox/en-us`

[9] ThreatExpert Ltd, Automated threat analysis, accessed Feb. 18, 2009 (Feb. 2009).
URL `www.threatexpert.com/`

[10] Symantec Corp., Understanding Heuristics: Symantec's Bloodhound Technology, Symantec White Paper Series XXXIV (1) (1997) 1–14.

[11] T. Lee, J. J. Mody, Behavioral Classification, in: Proceedings of EICAR, 2006.

[12] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, Automated Classification and Analysis of Internet Malware, in: Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection, Springer, 2007, pp. 178–197.

[13] M. Christodorescu, S. Jha, C. Kruegel, Mining specifications of malicious behavior, in: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, 2007, pp. 5–14.

[14] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: Malware Analysis via Hardware Virtualization Extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, ACM, New York, NY, USA, 2008, pp. 51–62.

[15] A. Moser, C. Kruegel, E. Kirda, Limits of Static Analysis for Malware Detection, in: Annual Computer Security Applications Conference (ACSAC), 2007.

[16] L. Cavallaro, P. Saxena, R. Sekar, On the limits of information flow techniques for malware analysis and containment, in: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer-Verlag, 2008, pp. 143–163.

[17] Software Engineering Institute, Carnegie Mellon University, CERT®Advisory CA-1992-02 Michelangelo PC Virus Warning, accessed Jul. 26, 2011 (Sep. 1997).
URL `www.cert.org/advisories/CA-1992-02.html`

[18] G. Tesauro, J. Kephart, G. Sorkin, Neural Networks for Computer Virus Recognition, IEEE Expert 11 (4) (1996) 5–6.

[19] J. O. Kephart, B. Arnold, Automatic extraction of computer virus signatures, in: Proceedings of the 4th Virus Bulletin International Conference, Virus Bulletin, Oxford, UK, 1994, pp. 178–184.

[20] W. Arnold, G. Tesauro, Automatically Generated Win32 Heuristic Virus Detection, VIRUS 51.

[21] W. W. Cohen, Fast Effective Rule Induction, in: Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, USA, 1995, pp. 115–123.

[22] V. Bontchev, Current status of the caro malware naming scheme, accessed Sept. 15, 2009.
URL `www.people.frisk-software.com/~bontchev/papers/pdfs/caroname.pdf`

[23] MathWorks, Inc., TreeBagger, accessed May 12, 2010 (2010).
URL `www.mathworks.com/help/toolbox/stats/treebagger.html`

[24] N. Rafiq, Y. Mao, Improving heuristics, Virus Bulletin (2008) 9–12.

[25] S. Treadwell, M. Zhou, A heuristic approach for detection of obfuscated malware, in: Intelligence and Security Informatics, IEEE, 2009, pp. 291–299.

[26] Microsoft Corporation, Microsoft portable executable and common object file format specification, accessed Dec. 23, 2009 (2008).
URL www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

[27] VX Heavens, Virus collection, accessed Apr. 15, 2010 (2010).
URL vx.netlux.org/vl.php

[28] manpagez, man page hexdump section 1, accessed Oct. 12, 2010 (2004).
URL www.manpagez.com/man/1/hexdump/

[29] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, S. Rogers, An Investigation of Malware Type Classification, in: Proceedings of 5th International Conference on Information-Warfare and Security, Academic Conferences Limited, England, 2010, pp. 398–406.

[30] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, S. Rogers, Malware Type Recognition and Cyber Situational Awareness, in: Proceedings of the IEEE Second International Conference on Social Computing, IEEE, 2010, pp. 938–943.

[31] MathWorks, Inc., MATLAB (2010).
URL www.mathworks.com

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The WEKA Data Mining Software: An Update, SIGKDD Explorations 11.1.

[33] M. Maloof (Ed.), Machine Learning and Data Mining for Computer Security, Springer, London, UK, 2006.

[34] J. Lobo, A. Jiménez-Valverde, R. Real, AUC: a misleading measure of the performance of predictive distribution models, Global Ecology and Biogeography 17 (2) (2008) 145–151.

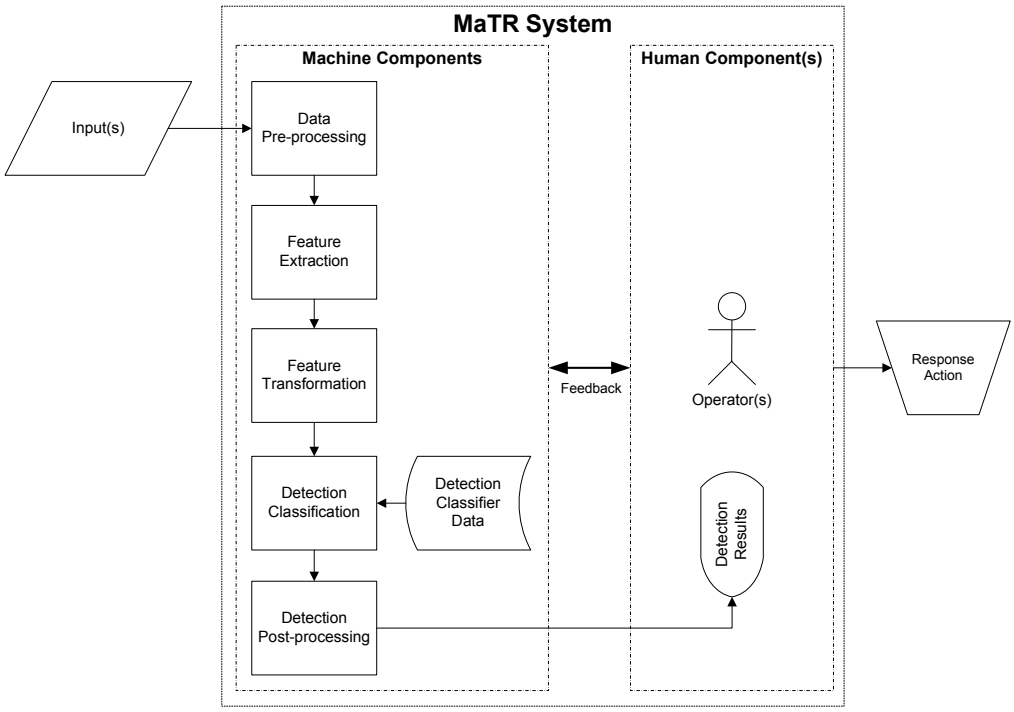

Figure 1: MaTR system process.

Table 1: Top seven n-grams across all folds.

0x00560001
0x56000100
0x72007900
0x00720079
0x0043006c
0x43006c00
0x44006500

Table 2: Mean AUC and confidence intervals.

Method	**Mean**	**95% CI**
MaTR	0.999914	0.999840 — 0.999987
KM retest	0.999173	0.998926 — 0.999421
KM original	0.9958	0.9934 — 0.9982

Table 3: Mean accuracy and confidence intervals.

Method	**Mean**	**95% CI**
MaTR	0.999166	0.999007 — 0.999325
KM retest	0.989919	0.988897 — 0.990941
Schultz (strings)	0.9711	*not reported*
Schultz (DLL function calls)	0.8936	*not reported*

Table 4: Mean confusion matrix for MaTR and KM n-gram retest.

Method	**TP**	**FP**	**TN**	**FN**
MaTR	3,112	2	2,517	3
KM retest	3,072	14	2,505	43

Table 5: Confidence intervals for FPR and FNR.

Method	**Mean**	**95% CI**
MaTR FPR	8.73e-4	5.80e-4 — 1.17e-3
MaTR FNR	8.03e-4	4.56e-4 — 1.15e-3
KM retest FPR	5.64e-3	3.65e-3 — 7.62e-3
KM retest FNR	1.37e-2	1.23e-2 — 1.51e-2
Schultz (strings) FPR	3.80e-2	*not reported*
Schultz (strings) FNR	2.73e-2	*not reported*
Schultz (DLL function calls) FPR	7.77e-2	*not reported*
Schultz (DLL function calls) FNR	2.89e-1	*not reported*

Table 6: Mean TPR and confidence intervals on unknown samples.

Method	**Mean**	**95% CI**
MaTR	0.985569	0.981508 — 0.989629
KM retest	0.949643	0.941552 — 0.957733
Antivirus 1	0.467606	0.449780 — 0.485431
Antivirus 2	0.388439	0.370665 — 0.406214
Antivirus 3	0.356071	0.340914 — 0.371229

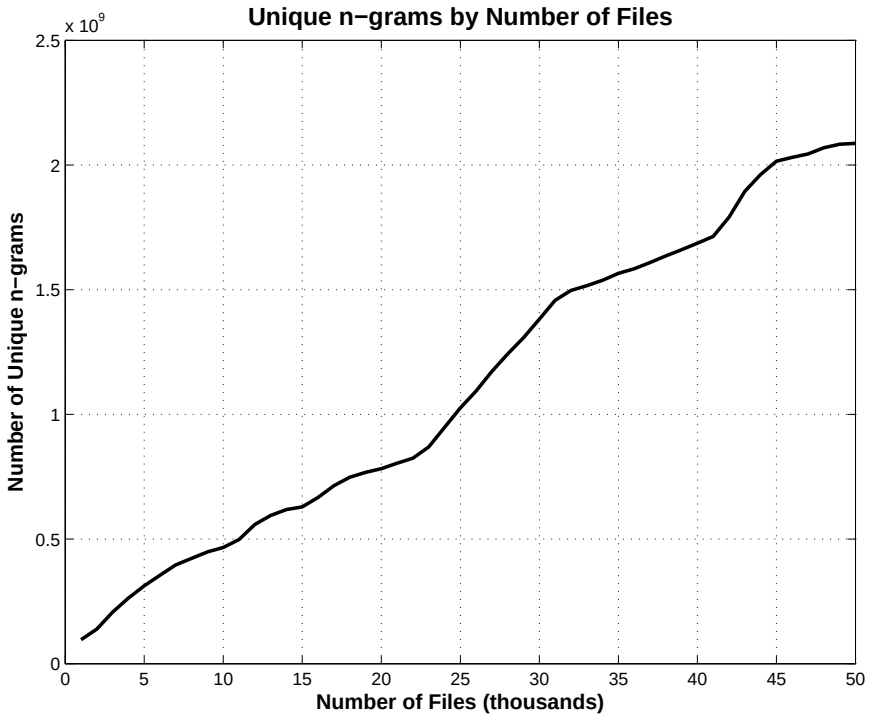

Figure 2: Number of unique n-grams increases linearly.

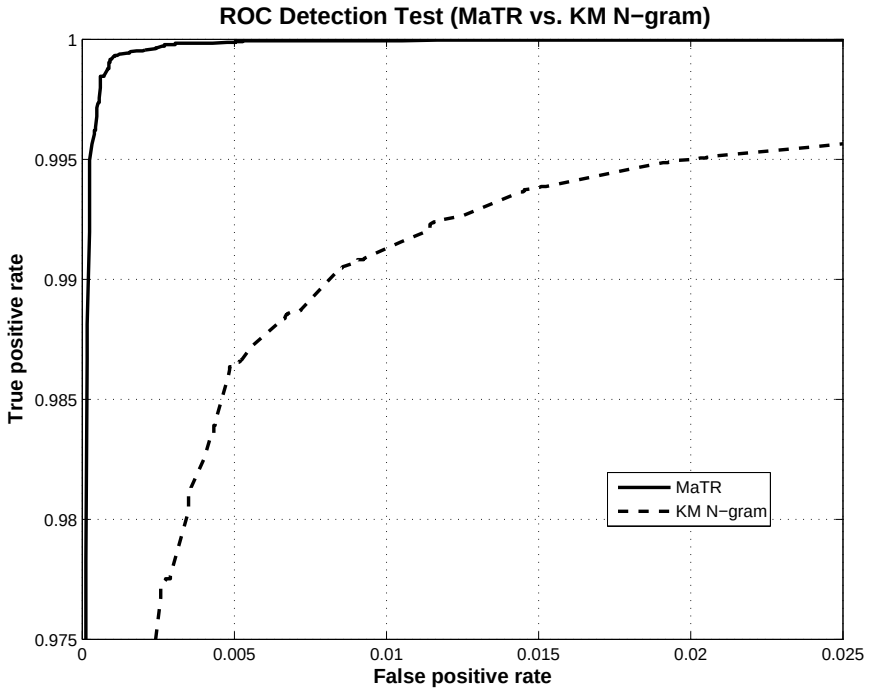

Figure 3: ROC curves for MaTR and KM n-gram retest.